

AD-A285 167



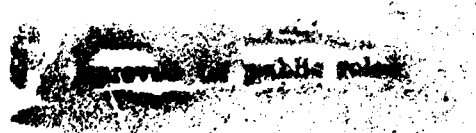
TASK: UU03  
CDRL: 05156  
March 1993

# Reuse Library Framework Version 4.1 User Manual

Informal Technical Data

DTIC  
ELECTE  
SEP 20 1994  
S a D

STARS-UC-05156/013/00  
March 1993



10788  
94-30888

94 3 26 024

**Best  
Available  
Copy**

TASK: U03  
CDRL: 05156  
March 1993

INFORMAL TECHNICAL REPORT  
For  
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS  
(STARS)

*RLF User's Manual, Version 4.1*

STARS-UC-05156/013/00  
March 1993

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031  
Delivery Order 0011

Prepared for:  
Electronic Systems Center  
Air Force Systems Command, USAF  
Hanscom AFB, MA 01731-5000

Prepared by:  
Paramax Systems Corporation  
12010 Sunrise Valley Drive  
Reston, VA 22091

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

**DTIC QUALITY INSPECTED 3**

Distribution Statement "A"  
per DoD Directive 5230.24  
Authorized for public release; Distribution is unlimited.

Data ID: STARS-UC-05156/013/00

**Distribution Statement "A"**  
**per DoD Directive 5230.24**

**Authorized for public release; Distribution is unlimited.**

Copyright 1992, Paramax Systems Corporation, Reston, Virginia  
Copyright is assigned to the U.S. Government, upon delivery thereto, in accordance with  
the DFAR Special Works Clause.

Developed by: Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

Permission to use, copy, modify, and comment on this document for purposes stated under Distribution "A" and without fee is hereby granted, provided that this notice appears in each whole or partial copy. This document retains Contractor indemnification to The Government regarding copyrights pursuant to the above referenced STARS contract. The Government disclaims all responsibility against liability, including costs and expenses for violation of proprietary rights, or copyrights arising out of the creation or use of this document.

In addition, the Government, Paramax, and its subcontractors disclaim all warranties with regard to this document, including all implied warranties of merchantability and fitness, and in no event shall the Government, Paramax, or its subcontractor(s) be liable for any special, indirect or consequential damages or any damages whatsoever resulting from the loss of use, data, or profits, whether in action of contract, negligence or other tortious action, arising in connection with the use or performance of this document.

TASK: U03  
CDRL: 05156  
March 1993

INFORMAL TECHNICAL REPORT  
RLF User's Manual, version 4.1

**Principal Author(s):**

---

*Del Gordon*

*Date*

---

*Kevin Russell*

*Date*

**Approvals:**

---

Task Manager *Richard E. Creps*

*Date*

*(Signatures on File)*

TASK: U03  
CDRL: 05156  
March 1993

INFORMAL TECHNICAL REPORT  
RLF User's Manual, Version 4.1

# Change Record:

<i>Data ID</i>	<i>Description of Change</i>	<i>Date</i>	<i>Approval</i>
STARS-UC-05156/013/00	RLF v.4.1 enhancements: PCTE integration enhancements; man pages.	March 1993	<i>on file</i>
STARS-UC-05156/006/00	RLF v.4.0 enhancements: Ported to SA-Motif; cascading menus; additional node types for greater bitmap specification flexibility; command-line options; .rlfrc initialization file; SNDL becomes LMDL, simpler modeling language; enhanced actions capability; <i>RLF GB User's Manual</i> becomes <i>RLF User's Manual</i> ; updated Usage Scenario sections; updated figures to Motif GUI	November 1992	<i>on file</i>
STARS-UC-04046/005/00	RLF GB 3.1 enhancements: Relationship View; Node History; N-Level Views, terminology updates; <i>RLF GB User's Manual</i> restructuring; new Usage Scenario sections	July 1992	<i>on file</i>
STARS-TC-04046/004/00	RLF 3.0 enhancements: actions, multiple file state; RLF GB enhancements to utilize RLF 3.0 features, menu <i>perestroika</i> , multiple inheritance features; object-oriented terminology	31 January 1992	<i>on file</i>
STARS-SC-03065/004/01	RLF GB enhancements; easier installation and startup procedures; customization section in appendix; discussion of multiple inheritance	04 October 1991	<i>on file</i>
STARS-SC-03065/004/00	Re-issued: Describes software upgrade to version 2.2	06 September 1991	<i>on file</i>
STARS-SS-004001/001/00	Original Issue	March 1991	<i>on file</i>

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED  
Informal Technical Report

4. TITLE AND SUBTITLE

*RLF User's Manual*

5. FUNDING NUMBERS

F19628-88-D-0031

6. AUTHOR(S)

Paramax Corporation

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Paramax Corporation  
1210 Sunrise Valley Drive  
Reston, VA 22090

8. PERFORMING ORGANIZATION  
REPORT NUMBER

STARS-UC-05156/013/00

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force  
Headquarter, Electronic Systems  
Hanscom AFB, MA 01731-5000

10. SPONSORING / MONITORING  
AGENCY REPORT NUMBER

05156

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution "A"

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This manual describes the use and basic customization of the Software Technology for Adaptable and Reliable Systems (STARS) Reuse Library Framework (RLF) and its primary user interface—the RLF Graphical Browser (GB); hereafter referred to as the RLF GB. The reader is not expected to be a programmer, but familiarity with the UNIX C shell, UNIX files and directories, and basic X Window System (X) interaction with some window manager is assumed. This manual assumes that X has been properly installed and that the user is able to start an X server.

14. SUBJECT TERMS

15. NUMBER OF PAGES  
95

16. PRICE CODE

17. SECURITY CLASSIFICATION  
OF REPORT  
Unclassified

18. SECURITY CLASSIFICATION  
OF THIS PAGE  
Unclassified

19. SECURITY CLASSIFICATION  
OF ABSTRACT  
Unclassified

20. LIMITATION OF ABSTRACT  
SAR



## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scope . . . . .	1
1.2	Identification . . . . .	1
1.3	Product Overview and Rationale . . . . .	2
1.4	RLF Fundamentals . . . . .	3
1.5	Notation Used in this Manual . . . . .	8
<b>2</b>	<b>Getting Started</b>	<b>9</b>
2.1	Assumptions . . . . .	9
2.2	Initial Setup Procedures . . . . .	9
2.3	Environment Variables . . . . .	10
2.3.1	The DISPLAY Environment Variable . . . . .	10
2.3.2	The RLF_LIBRARIES Environment Variable . . . . .	11
2.4	Invoking the RLF GB . . . . .	12
2.5	Usage Scenarios . . . . .	12
2.6	Summary of RLF GB Startup Procedures . . . . .	14
<b>3</b>	<b>Modes of Operation</b>	<b>15</b>
3.1	The Main Menu—Selecting RLF Libraries . . . . .	15
3.2	The Graph Display Window . . . . .	18
3.2.1	The Command Bar . . . . .	18
3.2.2	Graph Scrollbars . . . . .	18
3.3	The Graph Display . . . . .	20
3.4	The Filter View Command . . . . .	21
3.5	The Navigate View Command . . . . .	24
3.5.1	The Node History Command . . . . .	24
3.5.2	The Go To a Node Command . . . . .	25
3.5.3	The Go To Root Node Command . . . . .	26
3.6	The Search Command . . . . .	26
3.7	The Topology Command . . . . .	28
3.8	The Quit Command . . . . .	29
<b>4</b>	<b>Context-Sensitive Node Menus</b>	<b>31</b>
4.1	Context-Sensitive Node Menus—Specialization View . . . . .	31
4.2	Context-Sensitive Node Menus—Relationship View . . . . .	34
4.3	Canceling Pull-Down Menus . . . . .	35
4.4	Navigation . . . . .	35
4.4.1	Specialization View and Relationship View Navigation . . . . .	35
4.4.2	Going to a Child of a Node . . . . .	35
4.4.3	Going to a Parent of a Node . . . . .	36
4.4.4	Going to a Related Node . . . . .	37
4.5	The Relationship View . . . . .	39
4.5.1	Going to a Related Node in the Relationship View . . . . .	39

4.5.2	Going to a Referencing Node . . . . .	41
4.5.3	Going to a Referencing Category . . . . .	41
4.5.4	Going to a Referencing Object . . . . .	42
4.6	Going to Another Occurrence—Multiple Inheritance . . . . .	45
4.7	Centering a Node . . . . .	46
4.8	Performing Actions . . . . .	48
4.9	Inferencing—Getting Advice at a Node . . . . .	49
4.10	Asset Extract and Export Functions . . . . .	51
4.11	Node Suppression . . . . .	51
4.11.1	Suppressing Categories or Objects with No Descendants . . . . .	51
4.11.2	Unsuppressing Nodes . . . . .	52
<b>5</b>	<b>Arcs</b>	<b>53</b>
5.1	Displaying the Attributes of an Arc . . . . .	53
5.2	Suppressing Arcs from View . . . . .	54
<b>A</b>	<b>Appendix: Customization</b>	<b>55</b>
A.1	X Resources . . . . .	55
A.2	Bitmaps . . . . .	57
A.2.1	Bitmap File Naming Convention . . . . .	58
A.3	Fonts . . . . .	60
A.4	Invoking the Browser from a Shell Script . . . . .	60
A.5	Command Line Arguments for Setting X Resources . . . . .	60
<b>B</b>	<b>Appendix: Environment Variables</b>	<b>62</b>
<b>C</b>	<b>Appendix: Command Line Options</b>	<b>63</b>
<b>D</b>	<b>Appendix: The RLF Initialization File</b>	<b>65</b>
D.0.1	Library Instances . . . . .	66
D.0.2	Default Library . . . . .	66
D.0.3	Initial Category . . . . .	66
D.0.4	View Setting . . . . .	66
D.0.5	Bitmaps . . . . .	67
D.0.6	Library Advice Settings . . . . .	67
D.0.7	Specification Translator Settings . . . . .	68
D.0.8	The RLF Initialization File BNF . . . . .	68
<b>E</b>	<b>Appendix: Error Messages</b>	<b>72</b>
E.1	X Window System Errors . . . . .	72
E.2	OpenWindows Resources . . . . .	73
E.3	RLF Errors . . . . .	75
E.4	File Processing Errors . . . . .	77
<b>F</b>	<b>Appendix: Detailed Usage Scenarios</b>	<b>79</b>
F.1	Scenario 1: The Candidate Component is Known—Browsing . . . . .	79

F.2	Scenario 2: The Candidate Component is Unknown-Guided Search . . . . .	84
F.3	Scenario 3: Textual Query Mode-Querying Search Mechanisms . . . . .	88
<b>G</b>	<b>Appendix: PCTE</b>	<b>89</b>
G.1	Installing the PCTE Version of RLF . . . . .	89
G.1.1	Starting the PCTE Server . . . . .	90
G.1.2	Logging into PCTE . . . . .	90
G.1.3	Creating RLF Libraries in the PCTE Object Base . . . . .	91
G.1.4	Invoking the PCTE RLF Graphical Browser . . . . .	91
G.2	File Naming Restrictions . . . . .	92
<b>H</b>	<b>Appendix: Reporting Problems</b>	<b>93</b>
<b>I</b>	<b>Appendix: References</b>	<b>94</b>

**List of Figures**

1	Specialization and Individuation . . . . .	5
2	The Main Menu . . . . .	15
3	The Select a Library Menu . . . . .	16
4	The User Interface Components of a View . . . . .	18
5	The Elements of a Scrollbar . . . . .	19
6	Category and Object Nodes . . . . .	20
7	Category and Object Nodes with Actions or Inferencers Attached . . . . .	21
8	An Example of a Menu with Arrowhead Indicators for Submenus . . . . .	22
9	The Filter View Pull-Down Menu . . . . .	22
10	A Second-Level View Selected from the Filter View Pull-Down Menu . . . . .	23
11	The Navigate View Pull-Down Menu . . . . .	24
12	The Node History Menu . . . . .	25
13	The [Go To] Dialog Box . . . . .	26
14	The [Go To] Alert Box . . . . .	26
15	The [Search] Control Panel Menu . . . . .	27
16	The Topology Display . . . . .	28
17	The Quit Menu . . . . .	29
18	The Exit Alert Box . . . . .	29
19	Menu Option Sensitivity vs. Insensitivity . . . . .	31
20	A Context-Sensitive Category Menu . . . . .	33
21	Another Context-Sensitive Category Menu . . . . .	33
22	The [Go To a Child] Cascading Menu . . . . .	36
23	The [Go To a Parent] Cascading Menu . . . . .	37
24	The [Go To a Related Category] Cascading Menu . . . . .	38
25	A Relationship View for the <b>Example Binary Search</b> Object . . . . .	39
26	Related Categories/Objects Menu in a Relationship View for the <b>Example Binary Search</b> Object . . . . .	40
27	The [Go To a Referencing Category] Cascading Menu . . . . .	42
28	A Relationship View for a Category . . . . .	43
29	A Referencing Category Menu . . . . .	43
30	The [Search] Cascading Menu Showing Multiple Inheritance . . . . .	45
31	A Navigate Menu Containing the [Go To Other Occurrence] Option . . . . .	46
32	The [Perform Action...] Cascading Menu . . . . .	48
33	Performing an Action . . . . .	49
34	A Category Node with the [Advice] Menu Option . . . . .	49
35	The Components of an Advice Dialog Box . . . . .	50
36	The Alert Box from the [Advice] Menu Option . . . . .	50
37	The Arc Menu . . . . .	53
38	An Arc Attributes Display Window . . . . .	53
39	The Main Menu . . . . .	79
40	A Library Menu . . . . .	80
41	A Graph Display View . . . . .	80
42	A Category Node Menu . . . . .	81
43	Walking the Specialization Hierarchy . . . . .	82

44	An Object Node Menu . . . . .	82
45	Reusable Component Source Displayed in a Text Pager . . . . .	83
46	Selecting the [ <b>Extract source</b> ] Option from a Perform Action Menu . . . . .	84
47	The “Algorithms” Menu—Getting Advice . . . . .	85
48	An Example Initial Advice Dialog Box . . . . .	85
49	A Second Example Advice Dialog Box . . . . .	86
50	A Third Example Advice Dialog Box . . . . .	86
51	A Fourth Example Advice Dialog Box . . . . .	87
52	An Example Final Advice Dialog Box . . . . .	87
53	An Alert Box Displayed After an Inferencing Session . . . . .	87
54	A Search String Dialog Box . . . . .	88
55	A Search List Selections Menu . . . . .	89

## 1 Introduction

### 1.1 Scope

This manual describes the use and basic customization of the Software Technology for Adaptable and Reliable Systems (STARS) Reuse Library Framework (RLF) and its primary user interface—the RLF Graphical Browser (GB); hereafter referred to as the RLF GB. The reader is not expected to be a programmer, but familiarity with the UNIX C shell, UNIX files and directories, and basic X Window System (X) interaction with some window manager is assumed. This manual assumes that X has been properly installed and that the user is able to start an X server. This version of RLF can support execution using Portable Common Tools Environment (PCTE) as an underlying object management system. If you run the PCTE version, then certain guidelines must be followed when installing and running the software. PCTE-specific information has been gathered in Appendix G.

Some explanation of general RLF concepts is provided at the introductory level. This level of explanation should be adequate for the RLF end-user. Other RLF documents exist that contain more detailed information about the RLF. For information on constructing RLF libraries, consult the *RLF Modeler's Manual*. For detailed information on RLF administration issues, see the *RLF Administrator's Manual*. For help with installation of the RLF software, see the *RLF Installation Guide*. Since this version of RLF can support execution using Emeralde's implementation of PCTE as an underlying object management system, if the RLF is run with PCTE, it is assumed that the user understands PCTE and the Emeralde product, including the ability to construct *esh* scripts.

### 1.2 Identification

△ This manual is for the RLF Version 4.1 release, March, 1993. This release runs on SunOS, version 4.1.1 or later. This manual assumes a UNIX C shell interpreter is accessible to the user.

△ When building the RLF GB application from source code, the following software dependencies exist:

- STARS RLF, Version 4.1
- STARS Reusable Graphical Browser (RGB), Version 1.0
- SERC SA-Motif, Version 1.6 (commercial Ada bindings to X)
- OSF/Motif 1.1
- the X Window System, Release 4

△ and the following compiler dependency exists:

- Sun Ada Compiler, Copyright 1984, 1992 Sun-4 SunOS Release 4.0 and 4.1, Version 1.1(f)

If you do not already have the STARS products, you may obtain them by following the instructions in the *RLF Version Description Document* listed in Appendix I, "References."

At run-time, the only software dependency that exists is the following:

- the X Window System, Release 4

### 1.3 Product Overview and Rationale

Within a particular problem domain, applications tend to have similar characteristics. To facilitate reuse within a domain, it is useful to analyze the domain to identify the similarities and differences among applications, and to record the results of this analysis in the form of a domain model and generic domain architecture. A crucial requirement for domain-specific reuse libraries of the future is that they provide a means for storing and accessing this domain knowledge to readily support reuse-based application development.

The RLF was designed specifically to meet this need. The RLF supports the development and evolution of comprehensive domain models consisting of taxonomic information to describe domain structure and rule bases to capture heuristic domain knowledge. Such a domain model is capable of supporting many tools that impact all phases of the software lifecycle and support reuse of a broad spectrum of life cycle products beyond just source code.

The search and retrieval mechanisms of the RLF provide both novice and experienced users effective reuse library access. The RLF provides several modes of user interaction to support different search strategies. The fundamental mode of operation relies on a browsing paradigm, wherein users are presented with a graphical representation of the library domain model and move through the library under their own control, deciding which kinds of assets to investigate. The RLF also provides an advisor mode, which gives users rule-based advice about which assets to investigate next, and a query mode which allows users to submit database-style queries about library assets.

The main purpose of the RLF GB is to provide a graphical user interface to the RLF. This graphical interface is the primary user interface when the RLF is used as a domain-specific software reuse library tool. Another purpose of the RLF GB is to demonstrate large-scale Ada software reuse. The RLF GB itself reuses the RLF and RGB software subsystem components.

The RLF GB provides to the user a graphical, point-and-click interface to the RLF and its structured domain knowledge. The graphical interface to the RLF has completely replaced the textual interface of older versions of the RLF. The `LibraryEditor` application also has a graphical interface; the textual interface is no longer supported.

The RLF GB relies on the X Window System (X) to provide some of its capabilities, such as windowing, pull-down menus, graphics display, and mouse control. With these capabilities the RLF GB can graphically depict an RLF library, and the user can view and interact with the library using simple mouse actions. The library structure is also referred to as a *domain model*.

The functionality of this version of the RLF GB provides read-only access to the domain model. It does not provide any network creation or modification capabilities. The `Library_Editor` application provides some library modification capabilities, but is not discussed further in this manual. For a detailed discussion of the `Library_Editor`, see the *RLF Administrator's Manual*.

## 1.4 RLF Fundamentals

It is necessary to explain some RLF fundamentals to get the most benefit out of using the RLF GB. However, the scope of this manual places constraints on the depth of explanation; it must therefore remain an overview. See the RLF documents listed in Appendix I for more detailed information on the RLF.

The RLF consists of two primary subsystems, AdaKNET and AdaTAU. AdaKNET, which stands for "Ada Knowledge Network," is a knowledge representation system derived from the well-known artificial intelligence system KL-ONE. It enables the creation of knowledge structures known as "structured inheritance networks," which are also referred to as "semantic networks." Such networks can be used to model the characteristics of widely diverse domains, and are well suited to the construction of domain models supporting the development of software component libraries addressing specific application domains.

AdaTAU stands for "Ada Think, Act, and Update," which describes the basic functions of the AdaTAU rule-based inferencing subsystem. The AdaKNET and AdaTAU subsystems are unified in RLF through a software layer that enables AdaTAU inferencers to be associated with specific concepts in an AdaKNET network.

There are two central elements of an AdaKNET network: *category* and *object*. A *category* is a class of domain entities, such as "Reusable Software Component," and an *object* is an instance of a category, such as an actual file containing reusable source code—a particular sorting algorithm, for example, named "Quicksort." AdaKNET categories are organized hierarchically into what is called a *specialization hierarchy*. In this hierarchy, one category "specializes" another if the set of objects it represents is a subset of the set of objects represented by the other category; the specializing category is known as a subcategory, and it can specialize one or more categories. Objects are said to "individuate" categories, and each object individuates one or more category. The individuation relationship is similar in principle to the relationship between an Ada generic package and an instantiation of that package, or the relationship between a class of items such as "documents," and a particular item that belongs in that class or is a specific example of an item in that class, such as "the RLF User's Manual."



A category may have associated with it a set of *relationships* to other categories. These relationships, different from the specialization relationship described above, can be viewed as defining a set of arbitrary "relational attributes" between two categories. Subcategories are said to *inherit* all such attributes from their parent categories. That is, all the relational attributes defined for a category are also defined for its subcategories; the subcategories may have additional relational attributes of their own. The objects that individuate a particular category provide *fillers*, or values, (in the form of related objects) for those relational attributes, thus enabling relationships between objects to be represented. This relational attribute structure among categories and objects is said to form the *aggregation hierarchy* of the network. Relational attributes are typically used to indicate object interdependencies, "part-of" relationships (e.g., an object is a sub-component or part of another object), or derivation or history relationships (e.g., a code component is derived from a particular design component).

In addition, a category or object may have associated with it a set of text strings, data files, or integers. Each such item can be considered to be a *value attribute* of its associated category or object. This mechanism is used to associate actual component data, such as Ada source code files, with AdaKNET objects, and it is also used to record textual or integer information about categories or objects that is not readily representable solely within the AdaKNET specialization and aggregation formalisms. For example, a value attribute can be used to record the name and address of a person, an abstract for a document, or information instructing a user how to reuse a software component. Value attributes are attached to individual categories and objects and are not subject to the inheritance mechanism that is employed for relational attributes.

It is important at this point to clarify some terminology that is used throughout the RLF GB and throughout the remainder of this document. Relational attributes are generally referred to as "relationships," with a "related category" or "related object" as their target, whereas the value attributes are generally referred to simply as "attributes." Also, the term "concept" has been used historically within the RLF community to refer to either a category or an object, and that usage is continued herein. An "RLF network" (or "semantic network", or just "network") consists of a set of concepts and all the specialization and aggregation relationships between them. An "RLF library" consists of a network, the value attributes in the network, and any AdaTAU inferencers that may be associated with concepts in the network.

The RLF GB allows the user to traverse an RLF library via both the specialization and aggregation hierarchies (where the specialization hierarchy is assumed here to include objects for convenience) and also allows the user to access the value attributes and AdaTAU inferencers. Graphical views of the specialization hierarchies and the aggregation hierarchies are obtainable. The RLF GB draws a graph on the screen consisting of all the categories and objects in a network, depicted as the "nodes" of the graph, and connects the nodes with arcs representing the specialization and individuation relationships. An arc can be from a category node to another category node, representing the specialization relationship, or from a category node to an object node, representing the individuation relationship. These arcs

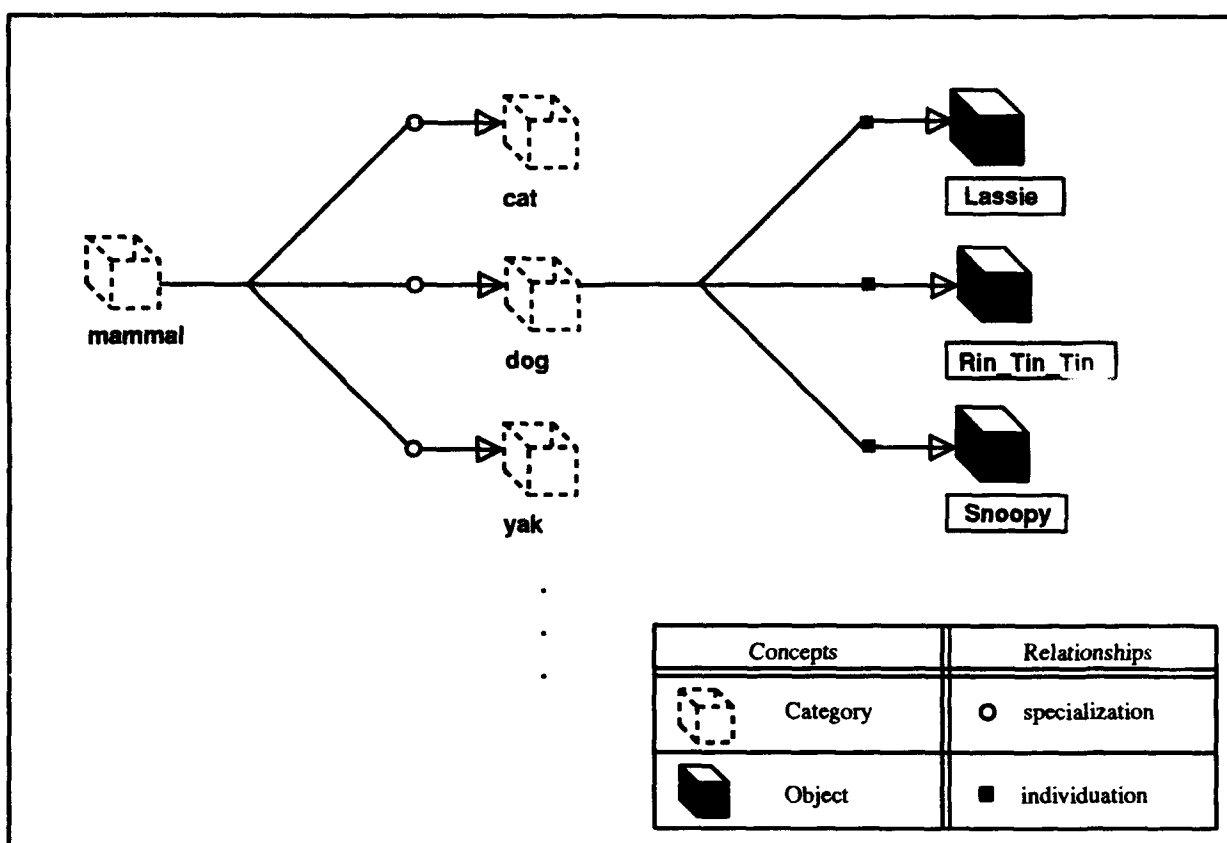


Figure 1: Specialization and Individuation

are directed arcs, so only one end of the arc has an arrowhead attached. The direction of the arcs displayed by the RLF GB flow from left to right. A node on the right specializes (or individuates) a node on its left when there is an arc between them. Figure 1 contains a partial graph that illustrates how the RLF GB depicts nodes and arcs. Within this graphical context, the aggregation hierarchy is traversed from a particular node by invoking functions from menus available at each node. These node menus contain the names of nodes related to that node, or options that produce a graphical representation of the aggregation hierarchy. Value attributes are accessed similarly, through the assistance of the RLF "action" mechanism described later in this manual, and AdaTAU inferencers are also accessed through a similar mechanism.

In the graph shown in Figure 1, the root of the graph is the category "mammal." The categories "dog" and "cat" specialize "mammal." If "mammal" has any relational attributes associated with it, then its specializations inherit those relationships. For example, if concept "mammal" possesses the relationship "is\_owned\_by," then both "dog" and "cat" also have the "is\_owned\_by" relationship. The object "Snoopy" individuates the category "dog", indicating there exists a concrete example in the world—a certain dog named "Snoopy"—of the abstract concept called "dog" in this domain of mammals. The filler for the "is\_owned\_by" relationship of "Snoopy" might be another object in the model called "Charlie Brown" (not shown in the figure).

As indicated earlier, concepts may have *actions* associated with them. Actions have a *name* and a *target*. The action defines an executable program, script, or procedure that can be invoked from the RLF GB. The target is a value attribute of the concept for which the action is defined. Most often, actions are defined for attributes whose values are stored in a disk file. An action could be defined, for example, that would run an Ada compiler and linker on a target source code file, and another action could be defined that would run the resulting executable file. Actions are inherited down the specialization hierarchy, via a mechanism similar to the inheritance mechanism previously described for relational attributes.

In an RLF model, since categories may specialize more than one category, *multiple inheritance* is possible. Multiple inheritance is a powerful inferential mechanism whereby a category may inherit relational attributes from more than one parent category. This allows a single category to embody the combined relationships of many parent categories. That category may then further refine those relationships by restricting the related category or reducing the number of fillers that is allowed. Through this mechanism, the commonalities among various categories are easily modeled, and a new category has to define only its unique features. This notion is also extended analogously to objects that individuate multiple categories; they possess (and can have fillers for) all the relationships that are defined within all their parent categories.

The method that the RLF GB currently uses to display an RLF specialization hierarchy, which can be represented visually by a construct known as a *directed acyclic graph* (DAG), is to unfold the graph into a simple tree structure (references to a "graph" elsewhere in this document refer to this tree representation of the graph). This allows the RLF GB to create aesthetically pleasing layouts, but causes a side effect that can sometimes be a source of confusion to the unwary user. As stated above, RLF provides a multiple inheritance mechanism, which allows concepts in a network to have more than one parent category. If the RLF GB were to display a DAG, rather than a tree, each concept in the network would be represented by a single node; if the concept had multiple parents, its corresponding node in the graph would have multiple arcs fanning in to meet it. However, in a network displayed as a tree, a concept is represented by multiple nodes if it has multiple parents, since a node in a tree can only have a *single* parent. Therefore, concepts with multiple parents appear as multiple nodes, each having only one arc leading to it. Each such arc emanates from a different parent, so that each instance of the concept within the tree provides an alternative context for understanding the concept. To minimize user confusion due to this approach, the RLF GB assigns a unique numeric suffix to the name of each node representing a multiply inherited concept. This gives every node in the graph a unique name, while also clearly identifying which nodes represent multiply inherited concepts. Note that concepts whose parents are categories with multiple parents will also appear multiple times in the tree, even if they have only one parent themselves.

Another note on terminology is appropriate at this point. An RLF GB graph consists of nodes, while an RLF specialization hierarchy (including objects) consists of concepts. Due to the fact that multiple nodes may represent a single concept, "node" and "concept" are not truly interchangeable terms. However, they are very nearly so, and they are used

*interchangeably within this document to a significant degree, when there is no danger of ambiguity.*

Operational details of the RLF GB are supplied in the following sections. First the Main Menu options are discussed, then the functions available in the command bar of every RLF GB graph display window are explained. Following that are the sections that describe the operation of the context-sensitive node menus. Usage scenarios can be found in the appendix.

## 1.5 Notation Used in this Manual

This manual describes procedures to interact with the UNIX operating system through a C shell interpreter, and procedures to interact with a graphical user interface. Therefore, examples are given that sometimes use a special notation. In presenting the examples, the following notation is used:

- **typewriter font**

This font represents information displayed by the computer. It is also used in code examples and textual passages to indicate use of the C shell command language or names of UNIX programs.

- *italic font*

This font is used in textual passages and code examples to indicate user-specified parameters (information you supply) for program names or command line options; it is also used to indicate special terms or phrases used in textual descriptions, and is also used in the conventional manner to place emphasis on words.

- **Escape**

Typewriter font enclosed in a box denotes a key on the keyboard.

- %

The percent sign is used to represent the C shell prompt.

- **[Quit]**

This bold font, enclosed in square brackets, represents a graphical command button, or menu option that is available from a graphical pull-down menu or cascading menu.

In the following example you would type the text "source ~/.cshrc" but you would not type the percent sign ("%"), which is the C shell prompt; also note that your C shell prompt may appear differently:

```
% source ~/.cshrc
```

In the next example you would insert the name of a text editor of your choice, such as *vi* or *emacs*, but you would type the filename ".cshrc" as shown:

```
% editor .cshrc
```

The following example shows the use of the square brackets to indicate a graphical command button. These buttons are displayed by X and must be clicked on with a mouse to be activated:

Click the **[Quit]** button to exit the RLF GB.

## 2 Getting Started

This section contains information needed to set up and invoke the RLF Graphical Browser. It should be of particular interest to first-time RLF users.

### 2.1 Assumptions

This manual assumes that the RLF GB application has been previously built and installed. The detailed instructions for the building and installation of the RLF GB can be found in the *RLF Installation Guide* document listed in Appendix I, "References."

Since the installation is assumed to have been previously performed, the RLF GB resource specification file (named "RLF\_Browser") should reside at the following location:

```
/usr/lib/X11/app-defaults/RLF_Browser
```

If the above file does not exist, then consult your system administrator. If it is not feasible or desirable to install the "RLF\_Browser" file at the above location at your site, then consult Appendix A, "Customization," of this document for alternative approaches.

This manual also assumes that you know how to start up the X Window System on your workstation. For more detailed information on starting and customizing X, see the X document listed in Appendix I, "References."

### 2.2 Initial Setup Procedures

If the RLF GB has already been built, then there will be an executable file named "Graphical\_Browser" and a C shell script named "RLF\_GB" available for execution. Place the pathname where the "Graphical\_Browser" and "RLF\_GB" files reside into your command search path. This is accomplished by modifying the value of the C shell `path` variable, either temporarily or permanently.

You can temporarily place the pathname of the "Graphical\_Browser" executable into your command search path by modifying the C shell `path` variable. This can be done by issuing the following command at your C shell prompt:

```
% set path = ( RLF_GB_pathname $path )
```

This will add *RLF\_GB\_pathname* to your current command search path, where *RLF\_GB\_pathname* is a pathname you supply. However, when your current shell is exited, this modification will be lost.

You can permanently place the pathname of the "Graphical\_Browser" executable into your command search path by editing the C shell initialization file (named ".cshrc") in your home directory, and inserting the following line:

```
set path = ( RLF_GB_pathname $path )
```

where `$path` is the shell variable that contains your current command search path, and `RLF_GB_pathname` is a pathname you provide to the directory that contains the "Graphical\_Browser" executable. Alternatively, you can add `RLF_GB_pathname` to any `set path` statement that may already exist in your `.cshrc` file. Since the C shell reads the `.cshrc` file every time it is started, these variable settings are effectively permanent until you edit the `.cshrc` again.

## 2.3 Environment Variables

There are two environment variables that must be set before the RLF GB can be run successfully. They are as follows:

- DISPLAY
- RLF\_LIBRARIES

You can edit the `.cshrc` file so that these variables are automatically set every time you invoke a new C shell.

(Actually, the above environment variables can be overridden or superseded with command-line options, which are discussed in the appendix. Thus, setting these environment variables is just one method among several for configuring the RLF GB.)

### 2.3.1 The DISPLAY Environment Variable

The DISPLAY environment variable is used by X to determine the host where your X server is running, as well as the number of the display to be used on that host. To have this environment variable set automatically each time you log in, insert the following line into your `.cshrc` file:

```
% setenv DISPLAY hostname:0
```

where the *hostname* is the name of your computer system as it is known to your network, and `:0` refers to the first display screen of your system (most conventional computer systems

are of the one-display type, but X allows applications to run on multi-display systems). You can obtain your host name, if you do not already know it, by issuing the UNIX command "hostname".

For example, if you issued the `hostname` command, as follows:

```
% hostname
```

and you received the following output:

```
sparc10
```

then you would set your `DISPLAY` environment variable as follows:

```
% setenv DISPLAY sparc10:0
```

### 2.3.2 The `RLF_LIBRARIES` Environment Variable

The `RLF_LIBRARIES` environment variable is used by the RLF to determine the location of the RLF libraries to be read. An RLF library must be created before the RLF GB can be used to browse that library. The procedures to create RLF libraries are explained in detail in the RLF documents listed in Appendix I, "References." To have this environment variable set automatically each time you log in, insert the following line into your `.cshrc` file:

```
setenv RLF_LIBRARIES rlf_library_pathname
```

where *rlf\_library\_pathname* is the pathname to a directory that contains at least one previously created RLF library.

The `source` command is a valuable tool for working in your current shell environment. When you execute the `source` command, your C shell reads and executes the commands in the specified file. Since no new subshell is created, you can use `source` to modify your current environment. Therefore, after editing your `.cshrc` file, you can then "source" it so that the new environment variables will be read by your current shell. This has an effect similar to exiting your current C shell and starting a new C shell. To "source" your `.cshrc` file, type the following command:

```
% source ~/.cshrc
```



where the tilde ( ~ ) is expanded by the C shell to be the path to your home directory.

If you need more information about setting up your X environment, consult the X documentation listed in Section I. Experienced users may wish to refer to Appendix A, "Customization."

## 2.4 Invoking the RLF GB

After the setup procedures have been completed, the RLF GB can then be started. A C shell script is provided for this purpose. The C shell is provided to check the values of the environment variables and the status of files that the RLF GB reads during execution. The RLF GB can be invoked without the use of this startup script; it is entirely optional. The RLF GB startup script is provided as a convenience to inexperienced RLF users in an attempt to automate some of the status checking that would be performed in a troubleshooting session.

The startup script is named "RLF\_GB". To invoke the RLF GB, execute the "RLF\_GB" script by typing its name at the C shell prompt and pressing the **Return** key:

```
% RLF_GB
```

After invoking the RLF GB you should see the Main Menu appear in a new window. Operating procedures for the RLF GB are explained in further detail in the following sections.

To invoke the RLF GB directly, without using the RLF GB startup script, type the name of the program on the command line:

```
% Graphical_Browser
```

As with the startup script, after invoking the RLF GB you should see the Main Menu appear in a new window. Operating procedures for the RLF GB are explained in further detail in the following sections.

## 2.5 Usage Scenarios

This section contains high-level descriptions of usage scenarios for the RLF. Since the program operations have not yet been discussed in detail, the low-level, detailed usage scenarios are deferred until the Appendix. See Appendix E, "Detailed Usage Scenarios," for the more detailed usage scenarios.

The RLF GB presents users with graphical views of an RLF library model and provides a variety of commands, both global and context-sensitive, that enable the user to navigate the

model and search for assets and to perform library-specific operations defined with the RLF action mechanism.

There are three basic search modes provided by the RLF GB to support different search strategies:

- The fundamental mode of operation relies on a browsing paradigm, wherein users navigate within the graphical views under their own control, deciding which kinds of assets to investigate.
- In addition, there is an advisor mode, which gives users rule-based advice about which assets to investigate next, based on information elicited from users concerning their needs.
- Also, a simple textual query mode is provided that allows users to submit queries about the names of library entities. (More sophisticated query capabilities are planned as future enhancements.)

Once interesting assets have been found, the RLF can be very useful for understanding and evaluating assets to determine if they are appropriate for the target system. The library model itself can greatly promote asset understanding through the wealth of descriptive and relational information that can be embodied directly in RLF models. In addition, the RLF action mechanism further promotes the asset understanding and evaluation process by enabling invocation of external tools that provide access to additional tool-specific asset information or that allow live, on-line testing of assets. Note that RLF libraries typically contain more than just code assets, and asset utilization can thus involve the reuse of other forms of assets, including various kinds of documents, test and evaluation materials, and so on. As mentioned previously, see Appendix C, "Detailed Usage Scenarios," for detailed, step-by-step usage scenarios.

## 2.6 Summary of RLF GB Startup Procedures

A synopsis of the RLF GB's startup procedures is given in the following list:

1. Start the X Window System.
2. Edit your `.cshrc` file to define the `path` shell variable, and the environment variables `DISPLAY` and `RLF_LIBRARIES`:

```
% editor ~/.cshrc
```

```
% source ~/.cshrc
```

3. Invoke the "RLF\_GB" script which runs the RLF GB executable:

```
% RLF_GB
```

or else invoke the RLF GB directly:

```
% Graphical_Browser
```

**Note:** The editing step need only be performed once initially; thereafter that step may be omitted.

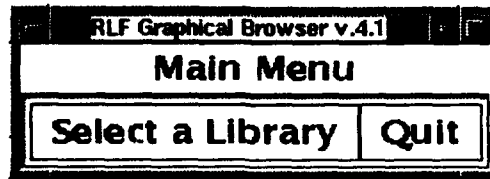


Figure 2: The Main Menu

### 3 Modes of Operation

#### 3.1 The Main Menu—Selecting RLF Libraries

The RLF GB displays its menus and windows graphically using X. The initial dimensions of the menus and windows can be specified in several different ways. In this release the dimensions are specified in the “*RLF\_Browser*” resource specification file. These dimensions can be modified to suit individual needs. See Appendix A, “Customization,” for further details on customizing the RLF GB.

When the RLF GB is invoked, the Main Menu is displayed (see Figure 2). The Main Menu provides options that allow you to either select an RLF library for viewing, or to exit the RLF GB application.

The Main Menu contains only two options: [**Select a Library**], and [**Quit**]. These options are displayed horizontally as buttons that you can “click” on (depress and release the mouse button). The horizontal area where the command buttons are displayed is referred to as the “command bar.” You move the mouse pointer (sometimes referred to as the “cursor”) over any command button using your mouse; the movement of the mouse is reflected by the movement of the pointer.

If you click on the [**Select a Library**] option, a menu is created that lists the available RLF libraries that have been stored in the directory identified by the *RLF\_LIBRARIES* environment variable. Consult the RLF documents listed in Appendix I, “References,” for guidance in creating RLF libraries. The Select a Library menu appears as a pull-down graphical menu, similar to the menu shown in Figure 3.

A pull-down menu, such as the Select a Library menu, will stay posted on the screen after it is invoked with a mouse click. You can then select one of the options available in the menu by moving the pointer over the desired option and clicking the mouse on that button. This is called *selecting* the option. You can dismiss the menu by clicking the mouse elsewhere on the screen, such as the menu’s titlebar, where the text “Main Menu” is displayed, or by pressing the **Escape** key on your keyboard.

For example, to select the “Ada Benchmarks” library, you place the pointer inside the box labeled “Ada Benchmarks.” Clicking the mouse button will cause the “Ada Benchmarks” library to be displayed.

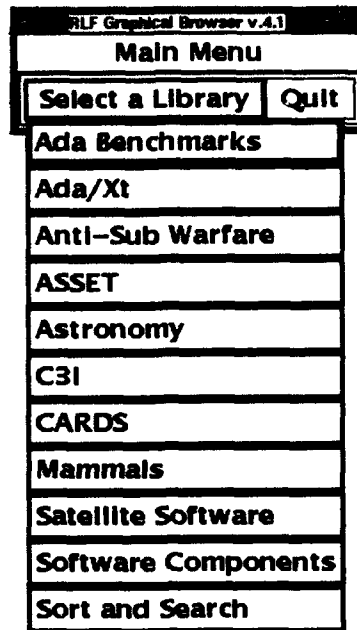


Figure 3: The Select a Library Menu

After you select a particular library, the RLF GB displays some informational statements regarding its processing status in the original window from which the application was invoked. These statements reflect the progress of the RLF GB's internal processing and can be safely ignored. They are displayed because large RLF libraries that contain hundreds of concepts can take several minutes to process; therefore, they indicate that the program is working properly. If the RLF GB does encounter a problem with its input or output processing, an exception will be raised and an error message will be displayed. A list of error messages and a description of their possible causes is given in Appendix D.

The status information appears similar to the following:

```
Welcome to the RLF Graphical Browser.  
Version 4.1
```

```
Copyright 1992, 1993, Paramax Systems Corp.
```

```
Main browser loop...
```

```
Getting the root node of the RLF network.  
Loading the current state of the RLF network.  
Creating the graphical browser's graph structures.
```

This graph contains 79 nodes.

Creating a full view.

Laying out static views.

After an RLF library is selected, the RLF GB creates a graphical depiction of the library in a new window, called a "view," or "graph display window." Multiple RLF libraries and views may be displayed simultaneously. After displaying the view of one RLF library, you can display the view of another RLF library by clicking on the **[Select a Library]** option from the graph display window's command bar. This will invoke the Select a Library menu again and you can choose another library at this point. Note that by displaying the first RLF library, the Main Menu window is expanded. Therefore, there is still only one RLF GB window being displayed at this point. Consult the X user guide listed in Appendix I, "References," for additional guidance in manipulating windows on your screen. The RLF GB graph display window and its operations are explained in more detail in the following sections.

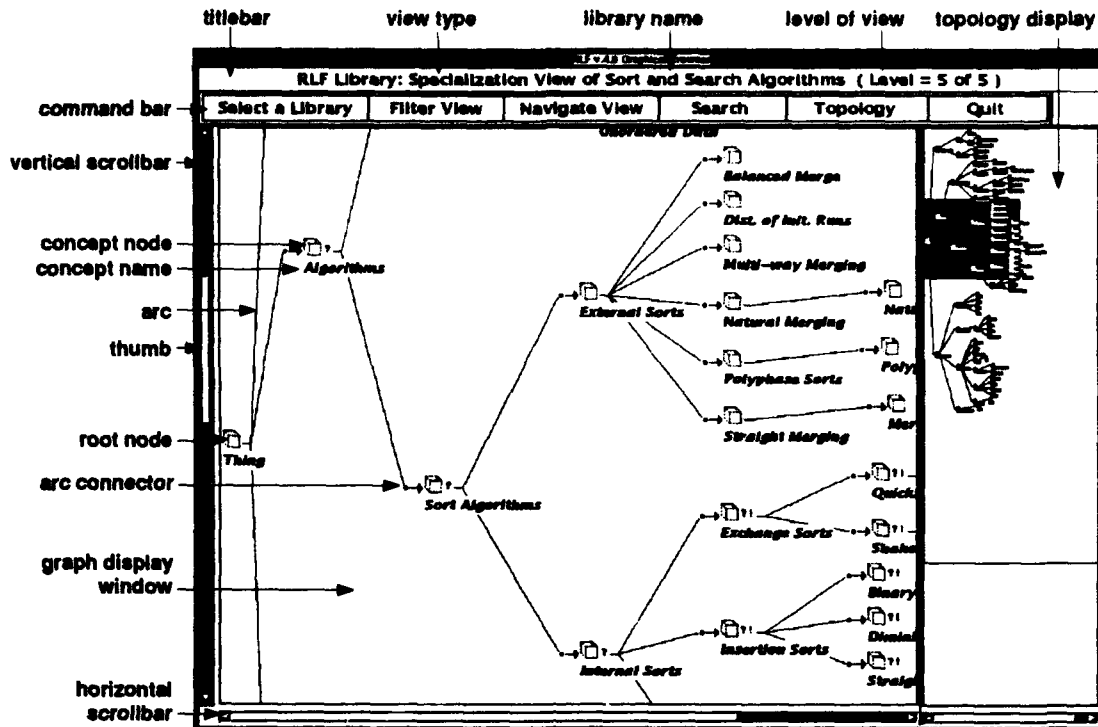


Figure 4: The User Interface Components of a View

### 3.2 The Graph Display Window

The user interface components of the graph display window are shown in Figure 4. The graph display window is also referred to as a "view." The example view in Figure 4 has the name of the corresponding RLF library displayed in its titlebar: "Sort and Search", along with other information that is described in the following sections.

#### 3.2.1 The Command Bar

Directly beneath the view's titlebar is a row of command buttons called the "command bar." Each command button is explained in detail in the following sections. Beneath the command bar is the graph display window, with a vertical scrollbar on the left side and a horizontal scrollbar at the bottom. The graph display window is where the graphical depiction of the RLF library appears.

#### 3.2.2 Graph Scrollbars

Graph display windows will have scrollbars when the graph is too large to fit into the available window display area in its entirety. Scrollbars can be either vertical or horizontal, and are used to control movement of the graph display in those directions. There is a sliding bar within a scrollbar, called the "thumb" (sometimes called "slider"), that moves within the

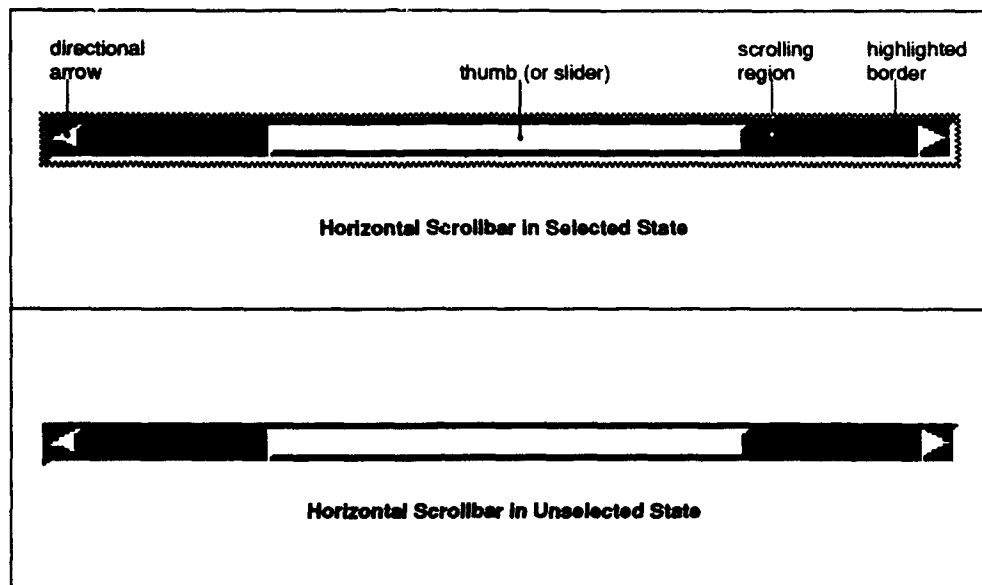


Figure 5: The Elements of a Scrollbar

scrolling region (see Figure 5). The thumb displays the position and the amount of the graph currently showing in the graph display window relative to the total size of the graph. If the whole graph fits within the given window, then no scrollbars are displayed.

When the pointer is positioned in the scrollbar and the mouse is clicked, the scrollbar becomes the focus of input. This state is indicated by a highlighted border (see Figure 5). Clicking the first (usually the left) or second (the middle) mouse button in the scrolling region causes the graph display window to scroll towards that side of the display. You can also *drag* the thumb in either direction by clicking and holding the first or second mouse button down while you move the mouse in either direction—left or right for horizontal scrollbars, or up or down for vertical scrollbars.

You can also use the *directional arrows* (see Figure 5) to move the scrollbar thumb in an incremental manner. By clicking on one of the directional arrows (sometimes called “incremental arrows”), you can cause the thumb and the graph display to move in that direction in small increments.

In addition to mouse control of scrollbars, you are also able to use the arrow keys on the numeric keypad of your keyboard to control the selected scrollbar. For example, if the vertical scrollbar is currently selected, then you can use the Up or Down keys (usually demarcated with arrows) to move the thumb in an incremental manner.



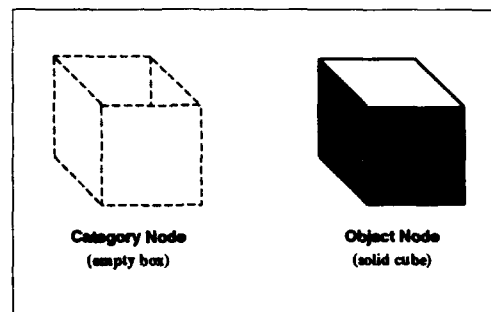


Figure 6: Category and Object Nodes

### 3.3 The Graph Display

In most graphs depicting an RLF semantic network, the graph will contain a root node, arcs, and two different types of nodes: category nodes and object nodes. Some networks may not contain any object nodes.

Category nodes are represented in the RLF GB by a bitmap that looks like a dotted, empty box. Object nodes are represented by a bitmap that looks like a solid cube with shading. See Figure 7 for an illustration of these bitmaps. These bitmaps can be customized to suit individual needs. The pathname to the `bitmaps` directory and individual bitmap filenames are specified in the `RLF_Browser` file. Alternative bitmaps can be specified by either modifying the pathnames to indicate different bitmap filenames, or by editing the existing bitmap files with the bitmap editor that accompanies X. See the X documentation for instructions on how to use the bitmap editor (the program name is `bitmap`). Also, see Appendix A, "Customization," for further details on customizing the RLF GB to satisfy individual needs.

There is also a finer granularity of detail available for specifying bitmap types. In addition to indicating whether a node is a category or an object, the bitmaps can represent what kinds of information may be available at that node. For example, a node may have a procedure attached to it (called an "action"), or an inferencer (these capabilities are explained in detail in the following sections), and it is possible to assign different bitmaps to nodes that contain differing sets of capabilities. Figure 6 shows the default set of bitmaps used to represent the informational content of the graph nodes.

Each node and each arc connector in the graph display is like a command button. When you place the pointer in the area over one of their bitmaps, they become highlighted. The highlighted state is indicated in one of two ways: either the bitmap is set to reverse video (white pixels become black, and vice versa), or a highlighting border is drawn around the bitmap. The highlighted state indicates that if you now click and hold the first (usually left) mouse button, you will invoke the menu for that particular node. Note that you must click *and hold down* (i.e., *drag*) the mouse to the right to invoke the Motif-style *cascading menus*. Cascading menus are also known as "pull-right" menus; they are called cascading menus because submenus are displayed to the right side of and slightly lower than the parent menu. Submenus are indicated with an arrowhead that appears in the menu entry (see Figure 8).

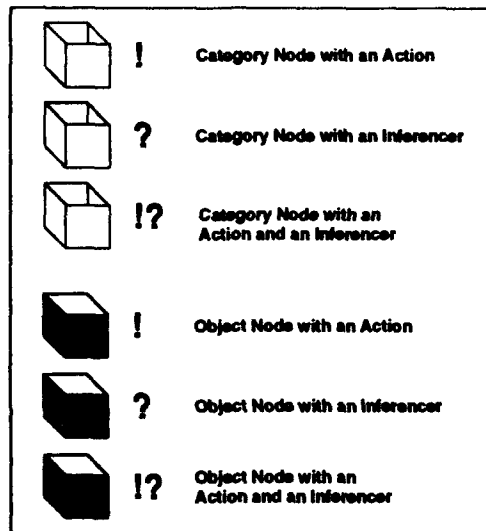


Figure 7: Category and Object Nodes with Actions or Inferencers Attached

You must drag the pointer to the right to invoke the cascading menus. Each node menu and its commands is explained in detail in the following sections.

There are two other notions associated with nodes in the graph display that are useful to library browsers: *visiting* and *focusing*. The “current focus” is defined as one particular node in a view. The currently focused node is the one that the RLF GB attempts to center in the view. Sometimes it is impossible to place the node of current focus exactly in the center of the view, since the node may be on one of the extreme edges of the view. In these cases, the RLF GB centers the node as near as possible with respect to one of the vertical or horizontal edges. The notion of “visiting” a node is necessary because a list is kept of the user’s course of navigation through the library. This list is available to the user as a command option. Therefore, some definition of what constitutes a visit is necessary. Those nodes that have been visited are added to the node history list. In the RLF GB, a visit to a node is defined as clicking on a node and invoking the node menu, even if the menu is subsequently canceled. Node history lists and node menus are described in detail in the following sections.

### 3.4 The Filter View Command

The [Filter View] commands allow you to display various subsets of the current graph. When you click on the [Filter View] button, a pull-down menu will be displayed. See Figure 9 for an illustration of the Filter View pull-down menu.

Various *levels* of graphs may be displayed. Clicking on the [Top-Level View] command creates a new view that contains only the top two levels—level 0 and level 1, where level 0 corresponds to the level of the root node—of the graph. Clicking on the [Second-Level View] command creates a new view that contains levels 0, 1, and 2. Additionally, you may specify an arbitrary level of graph to display, up to the maximum level of the graph.

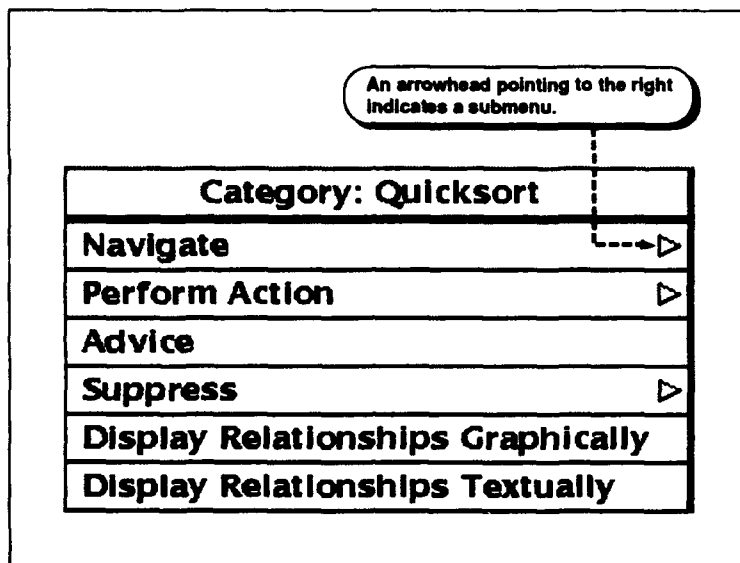


Figure 8: An Example of a Menu with Arrowhead Indicators for Submenus

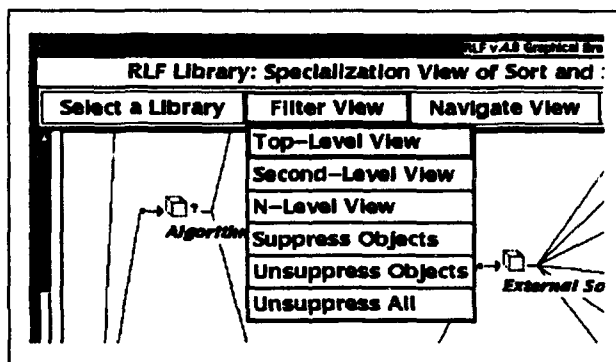


Figure 9: The Filter View Pull-Down Menu

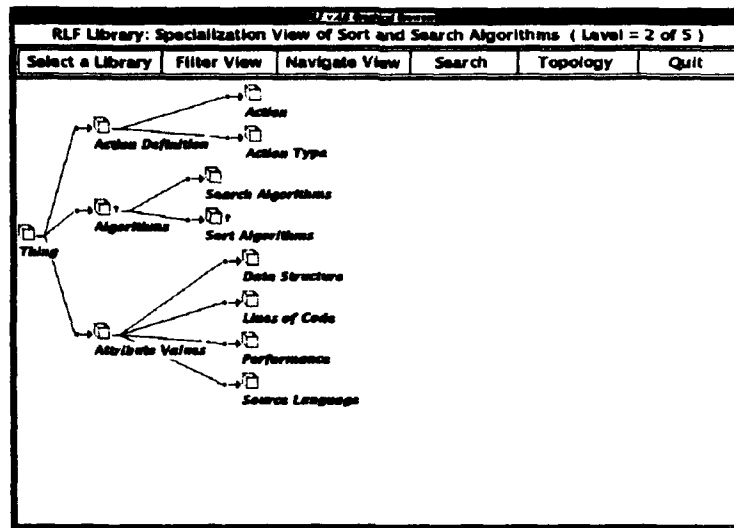


Figure 10: A Second-Level View Selected from the Filter View Pull-Down Menu

When you select the [N-Level View] command, a popup *dialog box* appears and you are prompted for the number of levels you wish to be displayed. See Figure 10 for an illustration of a Second-Level view.

A dialog box is used when the RLF GB needs to obtain some information from you, such as a number or some text, and usually contains a prompt, an input area, and command buttons that allow you to confirm or cancel the dialog input (see Figure 13 for a depiction of a dialog box).

Another way to filter the view is to limit the types of nodes that are displayed. If you want to see only categories, then click on the [Suppress Objects] subcommand button. The [Suppress Objects] command hides all the object nodes currently being displayed. The internal data structures that represent the graph are unaltered, thus the topology of the graph is not affected. This release of the RLF GB does not have the capability to dynamically re-layout the graph, however, so if you are trying to reduce the overall size of the graph, this command will not help. A dynamic re-layout capability may be provided in a future version of the RLF GB.

If, after suppressing objects, you want to see them again, then click the [Unsuppress Objects] subcommand button. This command causes all the object nodes that are currently hidden to become visible again, but only if their parent category is not hidden. The [Unsuppress All] command is available for the case where there might be more than just one type of node (or arc) suppressed, and it is a convenient way to restore the display of the graph back to its original state. Since nodes and arcs can be suppressed individually, the [Unsuppress All] command can be used to easily make them all visible once again.

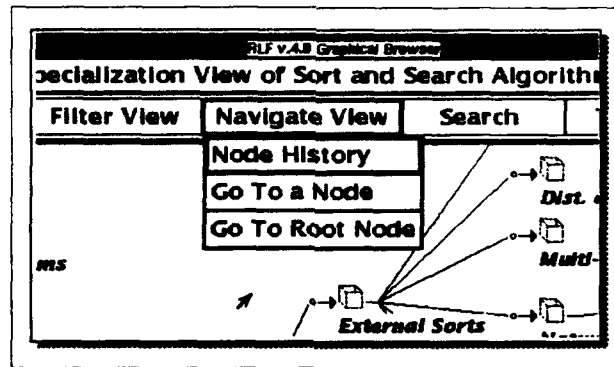


Figure 11: The Navigate View Pull-Down Menu

### 3.5 The Navigate View Command

The [Navigate View] command is used to pull down the Navigate View menu. This menu is shown in Figure 11. The [Navigate View] commands are used to traverse the graph when a specific destination is previously known. For example, if you want to go to the root node of the graph, or if you know the name of a specific node that you want to go to, then these commands can be used to move to that position of the graph. These commands are explained in further detail in the following sections.

#### 3.5.1 The Node History Command

Clicking on the [Node History] command button causes the Previously Visited Nodes menu to be displayed. A node history list of previously visited nodes is maintained by the RLF GB as you browse through the library. A "visit" is defined as clicking on a node and invoking its menu. The Previously Visited Nodes menu consists of concept name entries followed by a demarcation string to denote which type of view the node was visited in: a Specialization View, or a Relationship View; indicated by the strings "(S)" and "(R)" respectively. (These different types of views are explained in detail in the following sections.)

If you select one of the entries in the Node History menu, the graph display is focused upon that node. Focus is placed on the node in its corresponding window; i.e., if you select [Ada (R)], then the current focus goes to the "Ada" node in the Relationship View; if you select [Internal Sorts (S)], then the current focus goes to the "Internal Sorts" node in the Specialization View. Note that if one of these views happens to be occluded at the time you select the option, then the focusing and centering events may go unnoticed.

Nodes that have been most-recently visited are placed at the top of the menu; those nodes that have been in the list the longest are pushed towards the bottom of the list. See Figure 12 for an example of a Previously Visited Nodes menu.

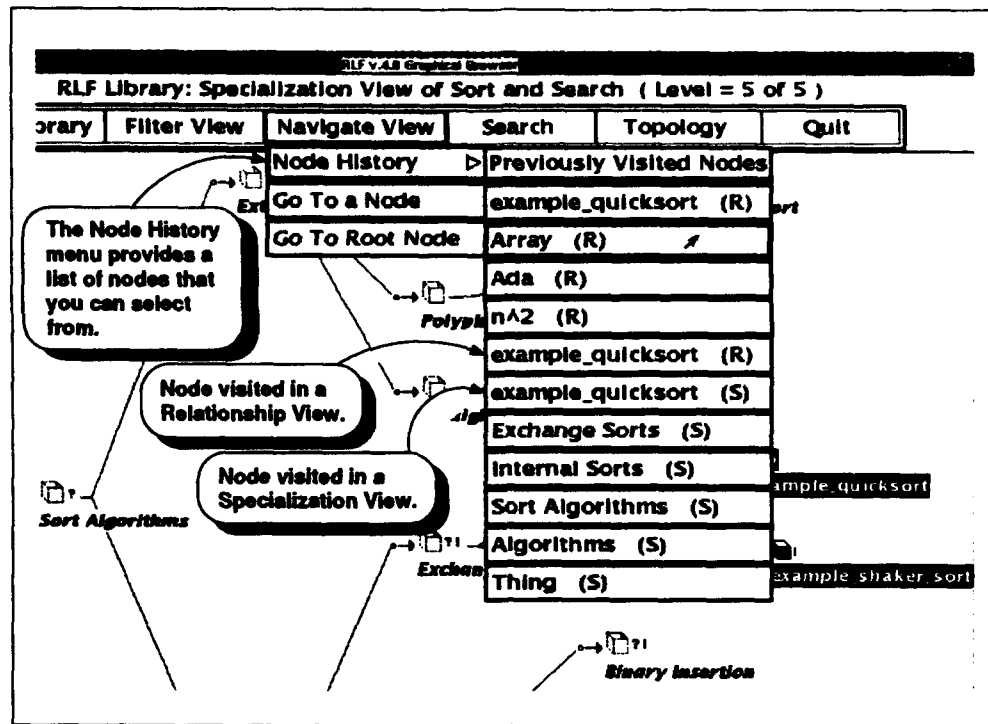


Figure 12: The Node History Menu

### 3.5.2 The Go To a Node Command

The [Go To a Node] command button allows you to specify a particular node name, then searches the graph for that node, and centers the view on that node. If you know the exact name of a certain node, this is a quick way to move to that node.

When you click on the [Go To a Node] command button, a dialog box is displayed (see Figure 13). The dialog box allows you to type the name of some node. Note that *you must place the screen pointer inside the text entry area and click the mouse before text can be entered*. The text entry point is indicated by a cursor that looks similar to a tall, skinny, uppercase 'I' (called an "I-beam" cursor) to show where new text will be inserted (you can think of 'I' as standing for "Insert"). When you are finished typing, you must click on the [OK] confirmation button or press the **Return** key for the RLF GB to accept your input and use it to search the graph. When the specified node is found, it is centered in the view.

If the text you type is erroneous, for example, the requested node name is nonexistent, then an *alert box* will be displayed (see Figure 14). The alert box is a notification that some error has occurred during processing, or that you have requested a function that cannot be performed at the time. The Go To alert box notification will include the node name that you typed so you can check it and make corrections.

Figure 14 shows the alert box that would be displayed if you requested the [Go To a Node] command to search for a node called "foobar" when a node of that name did not exist in the

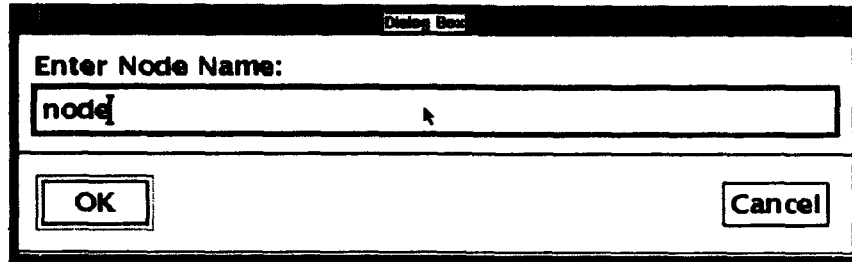


Figure 13: The [Go To] Dialog Box

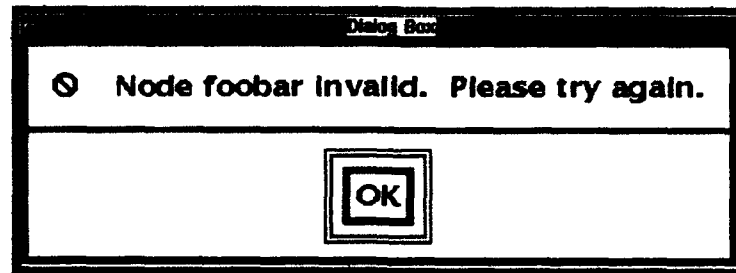


Figure 14: The [Go To] Alert Box

current library. This alert box informs you that the text you typed, "foobar", was invalid, and that you should try typing some different text. You must click on the [OK] confirmation button to acknowledge the alert box and continue.

### 3.5.3 The Go To Root Node Command

Clicking on the [Go To Root Node] command button causes the display of the graph to be centered (with respect to the top and bottom of the graph display window) on the root node. All graphs must have a single root node by definition. It is often desirable, after some initial browsing, to return to the root node of a graph when navigating through an unfamiliar library. The view is centered on the root node when it is initially opened, so this command is useful when it is desired to quickly return to the root node.

## 3.6 The Search Command

The [Search] command button allows you to enter a text pattern of at least three characters. The RLF GB then searches the current view for all concepts whose names contain that text, and displays a *control panel* with a list of the possible selections. The view is centered on the selected node. If you know the name of a certain concept, or a partial name, this is a convenient means of navigating through the library.

When you click on the [Search] command button a dialog box will be displayed with a prompt, "Enter Search String:". The dialog box allows you to specify the text of the search.

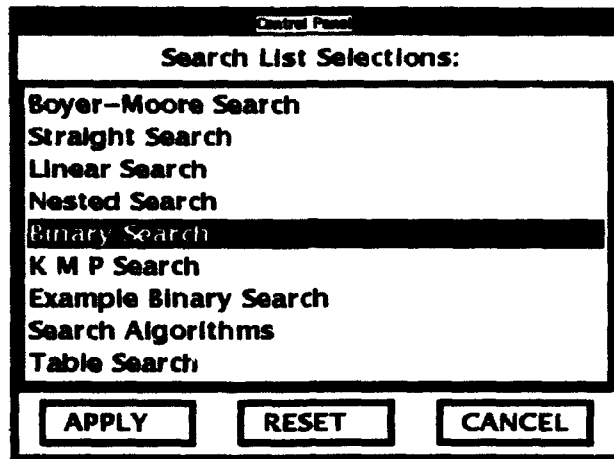


Figure 15: The [Search] Control Panel Menu

Note that you must click the pointer inside the text entry box before text can be entered. When you are finished typing, you must click on the [OK] confirmation button or press the Return key for the RLF GB to accept your input and use the text to search through the library for matches.

When the search is complete, a control panel menu will be displayed. You may then click on one of the menu options and the view will be centered over that node. See Figure 15 for an example of a Search List Selections control panel menu that was obtained by specifying "sea" as the search string.

Figure 15 shows a particular node name, "Binary Search," after it has been selected (clicked on with the mouse). Selecting a node name entry causes it to become highlighted. To traverse the graph to the selected node, click the [APPLY] button. To erase the current selection, click the [RESET] button. To cancel the entire Search operation, click the [CANCEL] button.

If you specify a search string that produces no matches, then an alert box is displayed with a message similar to the following:

No nodes contain the string foobar.

If you specify a search string that has less than three characters, then an alert box is displayed with this message:

I'm sorry, but you must enter at least three characters.



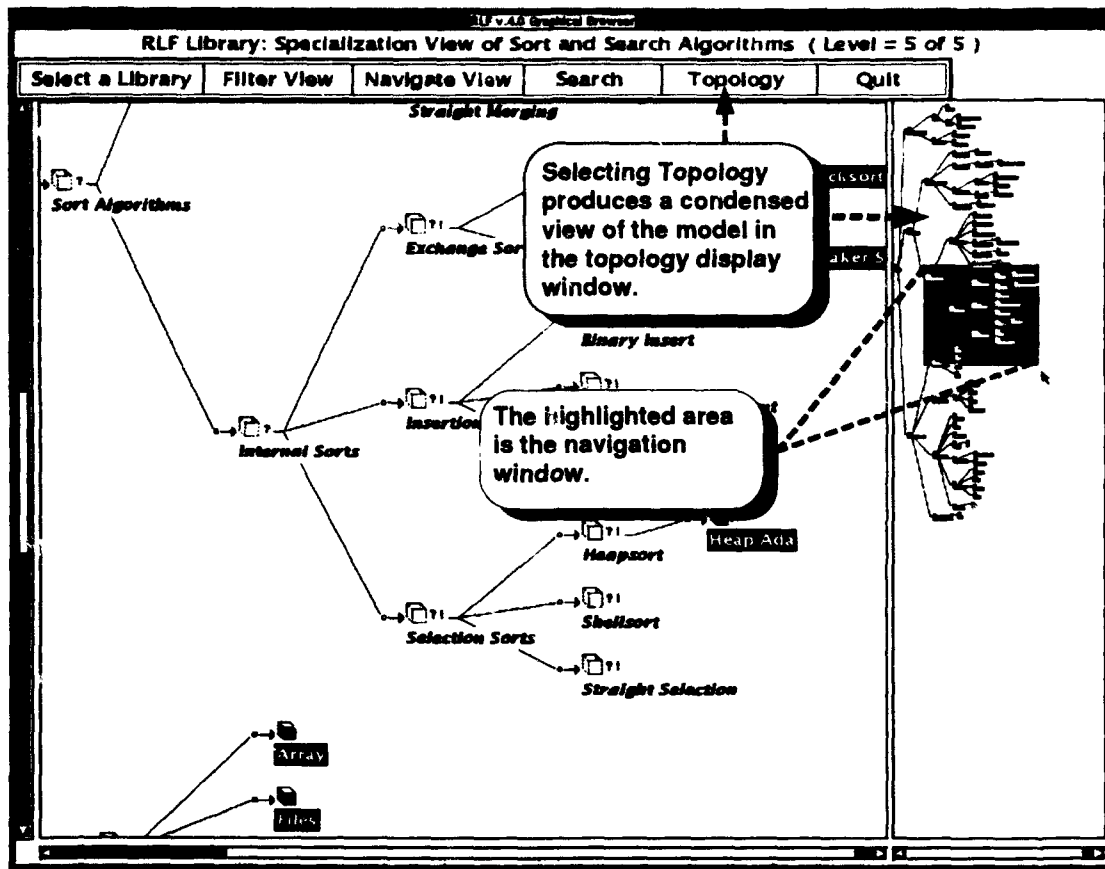


Figure 16: The Topology Display

### 3.7 The Topology Command

To obtain a topology display, or compressed view, of the currently displayed graph, the [Topology] command button can be clicked on. This command button provides a toggle switch for the topology display; that is, clicking on the command button repeatedly turns the display on and off. The topology display appears either on the right side or the bottom of the view, depending on the dimensions of the window. See Figure 16 for a depiction of a topology display.

The topology display contains a rectangular, highlighted region that acts similar to a scrollbar thumb (explained in Section 3.2.2). The highlighted region, called the "navigation window," indicates what portion of the graph is currently being displayed in the main graph display window. From this display you can get an idea of your current position relative to the remainder of the graph.

You can move the navigation window by clicking any mouse button anywhere within the topology display. The navigation window will move its center to the location where you click the pointer. Moving the navigation window causes the corresponding view to move, also. The new view closely corresponds to the navigation window's position within the topology

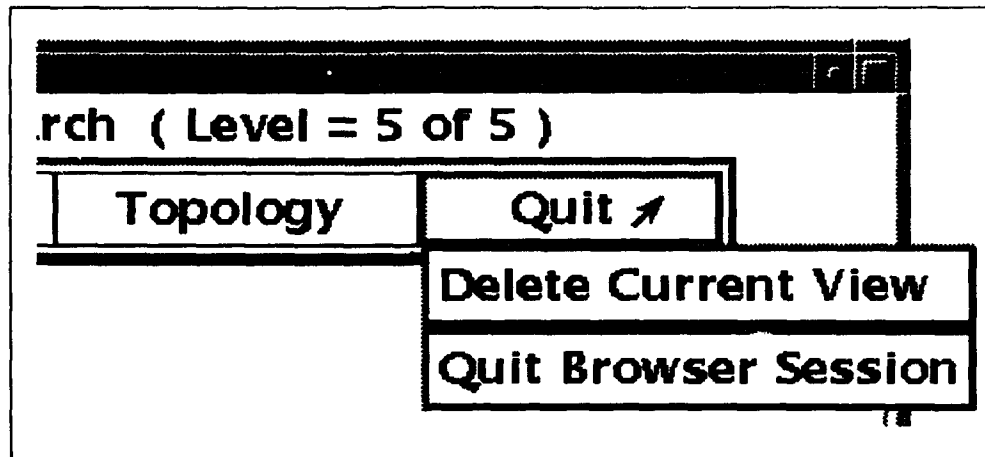


Figure 17: The Quit Menu

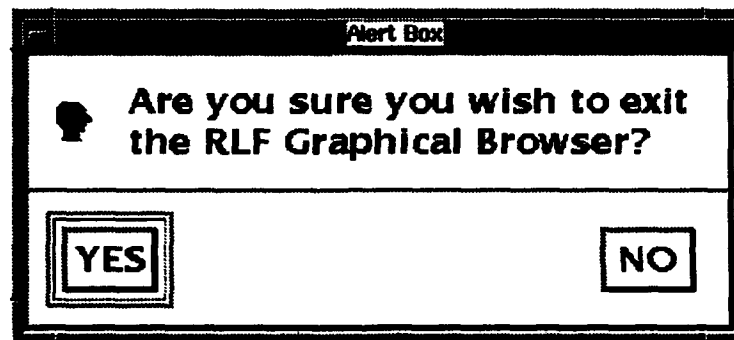


Figure 18: The Exit Alert Box

display.

If a graph is too large for even the topology display to fit all of it inside its display area, then the topology display itself will have scrollbars. By manipulating the display's scrollbar thumbs, you can view different areas of the topology display. The behavior of these scrollbars is analogous to the graph display window's scrollbars, explained in Section 3.2.2.

### 3.8 The Quit Command

Selecting the **[Quit]** command button causes the Quit menu to be displayed. The Quit menu is shown in Figure 17.

Selecting **[Delete Current View]** causes only the current view to be erased and exited. If it is the only view open, then the RLF GB will attempt to terminate completely. You will be prompted with an alert box, as shown in Figure 18, to confirm your intent to end the current session.

If there are other open views remaining after exiting the current view, then the RLF GB session remains open. From the command bar you can again select any command from its command bar, such as selecting another library to view, or you can exit the RLF GB completely by selecting the **[Quit Browser Session]** option from the Quit menu.

Selecting **[Quit Browser Session]** and acknowledging the exit alert box (clicking on the **OK** button) terminates the RLF GB session. When the RLF GB is terminated, all open views are closed and deleted, and you are returned to your original C shell.

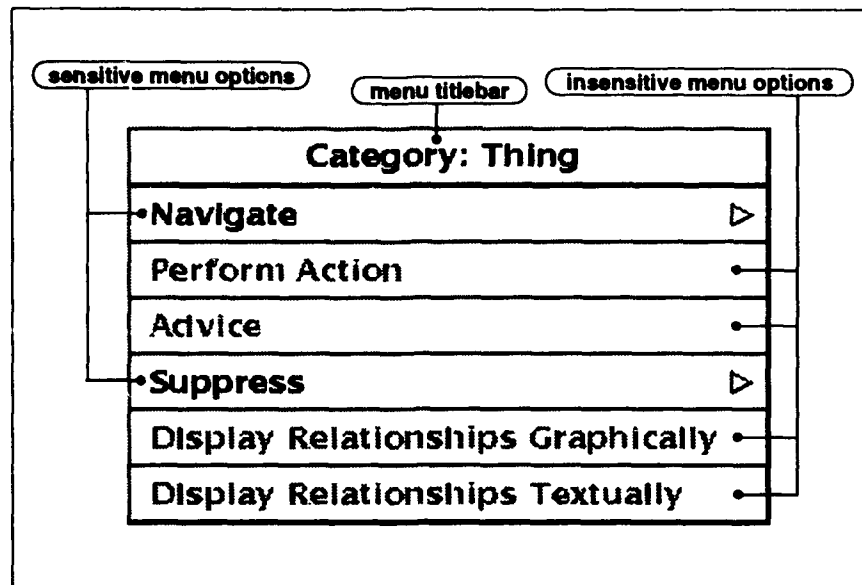


Figure 19: Menu Option Sensitivity vs. Insensitivity

## 4 Context-Sensitive Node Menus

### 4.1 Context-Sensitive Node Menus—Specialization View

When you click on the bitmap of a category or object node in a view, a context-sensitive menu is displayed near the screen location of the pointer. (The node bitmap behaves like a command button.) A context-sensitive menu may contain different options for different nodes depending upon what information has been associated with that particular node. An option of a node menu that is not available is made *insensitive* to mouse clicks. This means that the option will not respond to selection requests. The state of insensitivity is indicated by *graying-out* the text of the option. See Figure 19 for an example of a node menu that contains sensitive and insensitive options.

The possible menu options are given in the following lists. Some submenus that are displayed as a result of selecting a context-sensitive menu may not themselves be context-sensitive. Each menu option listed below is explained in detail in the following sections.

The *right arrowhead* symbol (“▷”) indicates that more functions are available for that particular option through cascading submenus. If you select an option with an arrowhead (click and hold down the mouse button, and then drag right), then a cascading submenu will be displayed that contains more options. To select an option from a submenu, you must drag the pointer into the submenu, position the pointer over the desired option, and then release the mouse button.

Every category or object node will contain the following options, though some subset of the options may be insensitive:

- **Navigate** ▸
- **Perform Action** ▸
- **Advice**
- **Suppress** ▸ (if the node has any descendants)
- **Suppress** (if the node has no descendants)
- **Display Relationships Textually**
- **Display Relationships Graphically**

However, the following subset of options will always be available (sensitive):

- **Navigate** ▸
- **Suppress** ▸ (if the node has any descendants)
- **Suppress** (if the node has no descendants)

The Navigate menu may contain any subset of the following options:

- **Go To a Child**
- **Go To a Parent**
- **Go To a Related Category** (if the node is a category)
- **Go To a Related Category/Object** (if the node is an object)
- **Go To a Referencing Category** (if the node is a category)
- **Go To a Referencing Object** (if the node is an object)
- **Go To Other Occurrence** (if the node has multiple parents)

and will contain one of the following options:

- **Center This Category** (if the node is a category)
- **Center This Object** (if the node is an object)

depending on whether the given function is valid in that context.

The Suppress menu will always contain the following options:

Category: Quadratic	
Navigate	▷
Perform Action	
Advice	
Suppress	▷
Display Relationships Graphically	
Display Relationships Textually	

Figure 20: A Context-Sensitive Category Menu

Category: Quicksort	
Navigate	▷
Perform Action	▷
Advice	
Suppress	▷
Display Relationships Graphically	
Display Relationships Textually	

Figure 21: Another Context-Sensitive Category Menu

- **Suppress**
- **Unsuppress Children**
- **Unsuppress Descendants**

An example of two different category menus is shown in Figure 20 and Figure 21. Note that for the two different category nodes, two different sets of available options (not grayed-out) are displayed, *i.e.*, the set of available options differs for each menu.

#### **4.2 Context-Sensitive Node Menus—Relationship View**

In Relationship Views, the context-sensitive node menus have slight variations from the menus in Specialization Views. In particular, the Navigation menu uses different terminology. This is necessary because the semantics of the navigational terms are different in a Relationship View. In a Specialization View, there are “parent-child” relationships, while in a Relationship View, there are “entity-relation” relationships, i.e., a relationship is defined as two entities that have a correspondence—a source node corresponds to its target node in some arbitrary manner defined by the library modeler. Therefore, in Relationship Views terms such as “related-referenced” and “source-target” are used in the Navigation menus to describe a structure that looks the same physically as the “parent-child” structure of the Specialization Views, but has a different meaning semantically.

In Relationship Views, the Navigate menu may contain any subset of the following options:

- **Go To a Referencing Category** ▷ (if the node is a category)
- **Go To a Referencing Object** ▷ (if the node is an object)
- **Go To a Referencing Node** ▷ (if the node is a category/object)
- **Go To a Related Category** (if the node is a category)
- **Go To a Related Object** (if the node is an object)
- **Go To a Related Node** (if the node is a category/object)
- **Go To Target** ▷ (if the node is a relation)
- **Go To Source** ▷ (if the node is a relation)
- **Go To Other Occurrence** ▷ (if the node has multiple occurrences)
- **Center This Relation** (if the node is a relation)
- **Center This Category** (if the node is a category)
- **Center This Object** (if the node is an object)
- **Center This Category in Specialization View** ▷ (if the node is a category)
- **Center This Object in Specialization View** ▷ (if the node is an object)

### 4.3 Canceling Pull-Down Menus

If you want to immediately cancel any pull-down menu, then you can do one of three things:

- click on the titlebar of the menu
- click on some other insensitive area
- press the Escape key on your keyboard

In the case of Figure 20, one way to dismiss the menu is to click inside the rectangular area that contains the text "Category: quadratic". This action will immediately cancel the menu without the RLF GB performing any further operations. The cancellation operation is useful when you just want to obtain information about nodes in the graph, but want to remain at your current location. The graph display contains a large quantity of information, but the menus are an important means of obtaining additional information about the contents of categories and objects in an RLF library.

### 4.4 Navigation

There are many different ways to navigate through an RLF library, and there are various navigation functions available through the context-sensitive, cascading node menus. Certain navigation methods are more appropriate than others, depending on the particular needs of the library browser at any given time. Once you become familiar with the different navigation methods that are available in the RLF GB, you should be able to choose the navigation function that is best suited to your situation. The navigation functions are explained in detail in the following sections.

#### 4.4.1 Specialization View and Relationship View Navigation

There are two different fundamental types of views—the Specialization View and the Relationship View—though the browsing methods for both views are very similar. The differences occur mainly in the semantics of the different terminology used for each respective view. The terminology and the various browsing functions are described in detail in the following sections.

#### 4.4.2 Going to a Child of a Node

One way to locate desired information in an RLF library is to walk the category hierarchy in a Specialization View. This means moving from parent to child categories in the hierarchy, going from more general concepts to more specific concepts in the domain, or vice versa. It also includes moving to or from objects of categories where objects exist. This navigational



<b>Category: External Sorts</b>		
<b>Navigate</b> ▷	<b>Navigate</b>	
<b>Perform Action</b>	<b>Go To a Child</b> ▷	<b>Children</b>
<b>Advice</b>	<b>Go To a Parent</b> ▷	<b>Balanced Merge</b>
<b>Suppress</b> ▷	<b>Go To a Related Category</b> ▷	<b>Dist. of Init. Runs</b>
<b>Display Relationships Graphically</b>	<b>Go To a Referencing Category</b>	<b>Multi-way Merging</b>
<b>Display Relationships Textually</b>	<b>Go To Other Occurrence</b>	<b>Natural Merging</b>
	<b>Center This Category</b>	<b>Polyphase Sorts</b>
		<b>Straight Merging</b>

Figure 22: The [Go To a Child] Cascading Menu

method is useful when you have some idea of the component you're searching for and the category hierarchy is structured in a way that tends to direct you towards desired components. For example, in a software components reuse library, if you were looking for an implementation of a binary search algorithm, you would know you were following an appropriate search path if you started walking the category hierarchy and found the following categories: **Algorithms** → **Sort Algorithms** → **Internal Sorts** → **Exchange Sorts** → **Quicksort**. At this point you would expect to find **Quicksort** examples nearby, such as the object **example\_quicksort**, as individuations of the **Quicksort** concept. To traverse the graph in this manner, choose the [Go To a Child] menu option.

When you choose the [Go To a Child] menu option by clicking on it, a cascading menu is displayed with a list of all the currently visible children of the selected category. If you select one of the menu options, the view will then be centered upon the chosen node. See Figure 22 for an illustration of the [Go To a Child] cascading menu. Note that an object node will never have this option, since an object node cannot have any children.

If the selected category does not have any children, then the [Go To a Child] menu option is not displayed.

#### 4.4.3 Going to a Parent of a Node

By going to a node's parent, you are walking the category hierarchy in the reverse direction—going from the more specific node to the more general node. By combining the navigation functions [Go To a Child] and [Go To a Parent] you can browse a library over a path based upon the nodes relevant to your current needs.

When you choose the [Go To a Parent] menu option by clicking on it, a cascading menu is displayed with a list of all the parent categories of the selected category or object. If you select one of the menu options, the view will then be centered upon the chosen category. See Figure 23 for an example of a Parents cascading menu. In this example, you are at the category node named **External Sorts** and that node only has one parent—the node named **Sort Algorithms**. By selecting the [Sort Algorithms] option, you are traversing the graph from a more specific category, **External Sorts**, to a more general category,

Category: External Sorts		
Navigate	▷	Navigate
Perform Action		Go To a Child
Advice		Go To a Parent
Suppress	▷	Go To a Related Category
Display Relationships Graphically		Go To a Referencing Category
Display Relationships Textually		Go To Other Occurrence
		Center This Category
		Parents
		Sort Algorithms

Figure 23: The [Go To a Parent] Cascading Menu

#### Sort Algorithms.

If the selected category or object does not have any parents, then the [Go To a Parent] menu option is insensitive.

#### 4.4.4 Going to a Related Node

Another useful way to browse an RLF library is to walk the relationship structure. This means moving between categories or objects via the relationships (see Section 1.4) that connect them. These relationships are orthogonal to the specialization and individuation relationships that are depicted graphically within a Specialization View and can be discovered by invoking the menus that allow these relationships to be traversed, or by producing a Relationship View, which is a graphical depiction of the relationship structure. A node related to the current node in this manner may be located anywhere in the graph, enabling easy, rapid traversal to other points of interest in the graph. Therefore, this navigational method provides a powerful mechanism to locate objects in the library that are related in ways that the user might not have originally known or considered.

One way to browse an RLF library through its relationship links is to choose the [Go To a Related Category] menu option from the Navigate node submenu. When you select this menu option a cascading menu is displayed with a list of all the related categories, if any, of the current category. If you select one of the menu options, the view will then be centered upon the category indicated by that selection. See Figure 24 for an illustration of the [Go To a Related Category] cascading menu.

For example, if the option [has\_avg\_case\_of of Performance] is selected, then the display will be centered on the concept **Performance**. In general, relationship clauses are displayed in the following format:

*relationship\_name of category\_name*

If the selected category does not have any relational attributes, then the [Go To a Related Category] option is not displayed in the menu.

Category: Sort Algorithms		
Navigate	▷	Navigate
Perform Action		Go To a Child ▷
Advice		Go To a Parent ▷
Suppress	▷	Go To a Related Category
Display Relationships Graphically		Go To a Referencing Category
Display Relationships Textually		Go To Other Occurrence
		Center This Category
		Related Categories
		has_avg_case_of of Performance
		has_best_case_of of Performance
		has_size_of of Lines of Code
		has_worst_case_of of Performance
		is_written_in of Source Language
		works_on of Data Structure

Figure 24: The [Go To a Related Category] Cascading Menu

There is a similar command available at object nodes—it is the [Go To a Related Category/Object] command. The only difference between the [Go To a Related Category] and the [Go To a Related Category/Object] command is that the latter provides a menu of destination nodes where some of the nodes are possibly objects instead of categories. The [Go To a Related Category/Object] command is only available from object nodes, while the [Go To a Related Category] command is only available from category nodes.

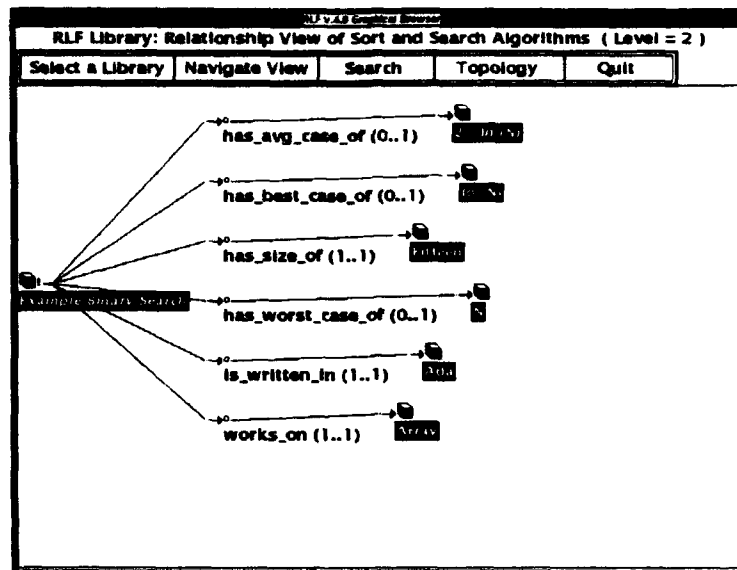


Figure 25: A Relationship View for the Example Binary Search Object

#### 4.5 The Relationship View

Another way to browse an RLF library through the relationship hierarchy is to choose the **[Display Relationships Graphically]** menu option from the node menu. (If the node does not have any relational attributes, then this option will not be available.) When you select this option, a Relationship View is constructed and displayed in a new window. An example of a Relationship View is shown in Figure 25.

A Relationship View is similar to a Specialization View in that it has mostly the same user interface elements such as command bar, scrollbars, title bar, and node menus. The main difference is in the terminology used to describe navigation operations. Since the Relationship View depicts a different kind of hierarchy, the "parent-child" structural metaphor of the Specialization View does not apply. The Relationship View shows the aggregation hierarchy (see Section 1.4), or how concepts are related to one another. Therefore, and as mentioned previously, to describe navigation operations in this kind of a hierarchy, terms such as "related-referenced," and "source-target," are used. Navigation in a Relationship View is described in the following sections.

##### 4.5.1 Going to a Related Node in the Relationship View

One Navigation option that is unique to Relationship Views is the ability to go to a connecting node, or, more precisely, to place the current focus upon a connecting node. In a Relationship View, the connecting nodes represent the names of the relationships that have been defined for that concept. For example, in Figure 25, the object Example Binary Search has the following relationships defined:

Object: Example Binary Search		
Navigate	D	Navigate
Perform Action	D	Go To a Child
Advice		Go To a Parent
Suppress		Go To a Related Category/O
Display Relationships Graphically		Go To a Referencing Object
Display Relationships Textually		Go To Other Occurrence
		Center This Object
		Related Categories/Objects
		has_avg_case_of of Logarithmic
		2 * ln (N)
		has_best_case_of of Logarithmic
		lg (N)
		has_size_of of Number
		Fifteen
		has_worst_case_of of Linear
		N
		is_written_in of Source Language
		Ada
		works_on of Data Structure
		Array

Figure 26: Related Categories/Objects Menu in a Relationship View for the Example Binary Search Object

```

has_avg_case_of (0..1)
has_best_case_of (0..1)
has_size_of (1..1)
has_worst_case_of (0..1)
is_written_in (1..1)
works_on (1..1)

```

These relationships have been defined by the library modeler. The numbers in parentheses represent the relationship's "cardinality." The cardinality indicates how many values the relationship can have. For example, the relationship `is_written_in (1..1)` indicates that the source node **Example Binary Search** can have only one language value as the target of the relationship—in this case, the target node is **Ada**. The relationship `has_avg_case_of (0..1)` indicates that the source node may or may not have an average case performance defined (zero or one). If a relationship does have a specific value defined, it is displayed in the Relationship View as a an object node such as **Ada** or **lg (N)** in the example cited above.

The Navigate menu of the object **Example Binary Search** in Figure 25 contains the [**Go To a Related Node**] option. If you select this option, a list of "relations" will be displayed in a cascading menu. See Figure 26 for the Related Categories/Objects menu that is displayed for the object **Example Binary Search**.

Selecting one of the relations displayed in the Related Categories/Objects menu causes the current focus to be placed upon the connecting node that represents that relation, and the view is centered upon the current focus.

Once you have traversed the Relationship View to a "relation" node, you may then invoke the Navigation menu there. The Navigate menus at "relation" nodes will offer one or both of the following options:

- **Go To Target**

- **Go To Source**

depending on whether or not it has both a "target" and a "source" node. In Figure 25, the relation node `is_written_in (1..1)` has both a "target" and a "source" node. Its target node is `Ada` and its source node is `Example Binary Search`. Selecting one of the `[Go To Target]` or `[Go To Source]` options causes the current focus to be placed on the appropriate node, and the view is centered upon the current focus.

#### 4.5.2 Going to a Referencing Node

The `[Go To a Referencing Node]` option is the inverse of the `[Go To a Related Node]` option described in the previous section. This option allows you to walk "up" the tree to a relation that references the node of current focus.

For example, in Figure 25, if you selected the `[Go To a Referencing Node]` option from the Navigation menu of node `Ada`, then the Referencing Categories/Objects menu is displayed that offers the option: `[is_written_in (1..1)]`. If you select this option, the focus will be placed upon that relation.

#### 4.5.3 Going to a Referencing Category

The `[Go To a Referencing Category]` option is the inverse operation of the `[Go To a Related Category]` option. For example, if you chose to go to the category `Quadratic` from the category `Quicksort` by traversing the relationship hierarchy with the `[Go To a Related Category]` option of the Navigation menu, then the category `Quadratic` will have the category `Quicksort` stored as a referencing category. This option is useful to return to the original category if you traverse the graph and then determine that the destination node was not what you were looking for, or to discover what other concepts in the network reference the current category with some relationship. To traverse the graph in this manner, choose the `[Go To a Referencing Category]` menu option. Remember that you can always cancel a menu by clicking its titlebar or pressing the `Escape` key. Therefore, you can use the menus to obtain information only, and remain at your present location.

When you choose the `[Go To a Referencing Category]` menu option by clicking on it, a cascading menu is displayed with a list of all the categories which reference the selected category. The menu options also include the name of the relationship through which the selected category was referenced. If you select one of the menu options, the view will then be centered upon the referencing category. See Figure 27 for an illustration of a Referencing Categories cascading menu.

For example, if the option `[has_worst_case_of : Quicksort]` is selected, then the display will be centered on the concept `Quicksort`. Since this menu was invoked from the `Quadratic` category, the example option can be interpreted as saying, "The Quicksort algorithm has a

Category: Quadratic		
Navigate	▷	Navigate
Perform Action		Go To a Child ▷
Advice		Go To a Parent ▷
Suppress	▷	Go To a Related Category
Display Relationships Graphically		Go To a Referencing Category ▷
Display Relationships Textually		Go To Other Occurrence
		Center This Category
		Referencing Categories
		has_worst_case_of : Shakersort
		has_avg_case_of : Shakersort
		has_worst_case_of : Quicksort

Figure 27: The [Go To a Referencing Category] Cascading Menu

worst-case Performance that is Quadratic.” In general, referencing relationship clauses are displayed in the following format:

*relationship\_name : referencing\_category\_name*

If the selected category is not referenced by any other categories, then the [Go To a Referencing Category] option is insensitive.

In a Relationship View, selecting the [Go To a Referencing Category] option has the same effect—the graph is traversed from the current node to the node that references the current node as an attribute. The main difference in the Relationship View from the Specialization View is that this relationship link is visible in the graphical display because they are displayed as arcs. In Figure 28, a Relationship Display is shown for the category **Binary Search**. If you click on node **Performance [2]** and select [Go To a Referencing Category] from the Navigate menu, the Referencing Category menu appears, as shown in Figure 29. In this case, selecting the option [**Binary Search via has\_avg\_case\_of (1.1)**] will cause the current focus to be placed upon the **Binary Search** node. This is similar to traversing the “child-parent” links in the Specialization View, except instead of following specialization links there, you are following relationship links here in a Relationship View.

#### 4.5.4 Going to a Referencing Object

The [Go To a Referencing Object] option is the inverse operation of the [Go To a Related Category/Object] option and is analogous to the [Go To a Referencing Category] command described in the previous section; the only difference is that this command is available for object nodes instead of category nodes. This command is useful to return to the original object if you traverse the graph and then determine that the destination node was not what you were looking for, or to discover what other concepts in the network reference the current object via relationships. For example, this command could be used to determine which software components are written in Ada if there were a `is_written_in` relationship defined. To traverse the graph in this manner, choose the [Go To a Referencing Object] menu option. Remember that you can always cancel a menu, therefore, you can use the menus to obtain information only, and remain at your present location.

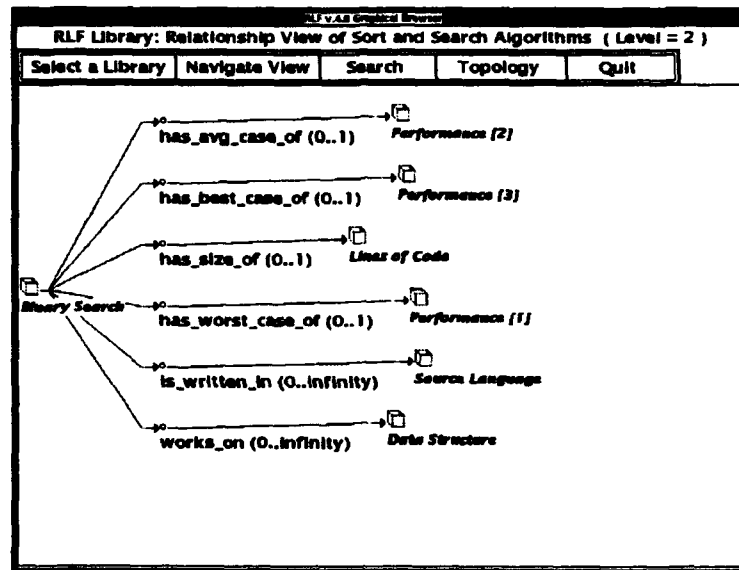


Figure 28: A Relationship View for a Category

Category: Performance [2]		
Navigate	Navigate	
Perform Action	Go To a Related Category	
Advice	Go To a Related Node	
Suppress	Go To a Referencing Category	Referencing Category
	Go To a Referencing Node	Binary Search via has_avg_case_of (0..1)
	Go To Other Occurrence	
	Center This Category	
	Center This Category in Specialization View	

Figure 29: A Referencing Category Menu



When you select the **[Go To a Referencing Object]** menu option, a cascading menu is displayed with a list of all the objects and/or categories which reference the current object. The menu options also include the name of the relationship through which the current object was referenced. If you select one of the menu options, the view will then be centered upon the referencing object.

For example, if the option **[is\_written\_in : example\_quicksort]** is selected, then the display will be centered on the object **example\_quicksort**. In general, referencing relationship clauses are displayed in the following format:

*relationship\_name : referencing\_node\_name*

If the selected object is not referenced by any other objects, then the **[Go To a Referencing Object]** option is insensitive.

In a Relationship View, the **[Go To a Referencing Object]** option functions the same as the **[Go To a Referencing Category]** option described in the previous section. The only difference is that the related nodes are objects instead of categories.

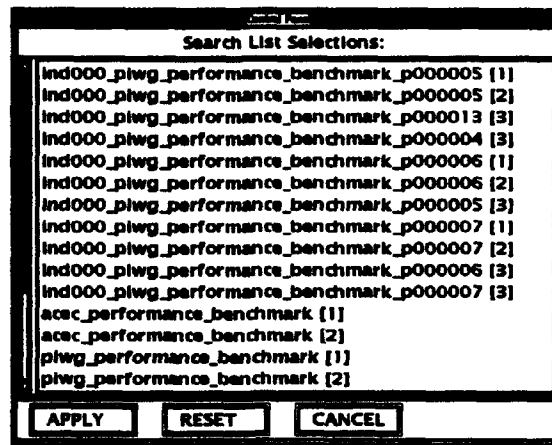


Figure 30: The [Search] Cascading Menu Showing Multiple Inheritance

#### 4.6 Going to Another Occurrence—Multiple Inheritance

If a category or object has more than one parent in the RLF semantic network, then it will appear as multiple nodes in the graph. The [Go To Other Occurrence] option enables you to quickly and easily move within the graph to a node representing one of the other occurrences of a category or object. This allows you to view a category or object in an alternative context and therefore may provide an enhanced understanding of that category or object.

As was previously explained in Section 1.4, due to the mechanism of multiple inheritance any given category or object in an RLF semantic network may have more than one parent, and thus may appear as multiple nodes in the graph that is displayed to the user. Therefore, such nodes have a number appended to their names to uniquely identify them. The numbers start with 1 and are assigned in the order in which the nodes are created during graph creation. The number is enclosed in square brackets and appended to the category or object name. For example, if the category `piwg_performance_benchmark` had more than one occurrence, then the first occurrence would be labeled `piwg_performance_benchmark [1]`, the second occurrence would be labeled `piwg_performance_benchmark [2]`, and so on. All such nodes will have a [Go To Other Occurrence] option displayed in their menus.

An example of a menu displayed as a result of running the [Search] command with the string `benchmark` when multiple inheritance exists is shown in Figure 30. This menu shows that when a category or object has more than one parent it will have more than one occurrence in the view.

For example, in Figure 30, there are names of categories and objects that occur more than once, such as `accec_performance_benchmark` and `piwg_performance_benchmark`. Each of these two categories appears twice, but each occurrence is given a unique number to differentiate them. Selecting different occurrences of the same category name in the menu causes the view to be centered on the respective occurrences of that category in the view.

Category: piwg_performance_benchmark [1]		
Navigate	Navigate	
Perform Action	Go To a Child	
Advice	Go To a Parent	
Suppress	Go To a Related Category	
Display Relationships Graphically	Go To a Referencing Category	
Display Relationships Textually	Go To Other Occurrence	Other Occurrences
	Center This Category	piwg_performance_benchmark [2]

Figure 31: A Navigate Menu Containing the [Go To Other Occurrence] Option

Figure 30 shows a different type of user interface component—a control panel—as opposed to a pull-down or cascading menu. To select an option from a control panel, you must choose an item from the list by clicking on it and then click on the **APPLY** button. This is the same type of interface as the Search List Selections control panel previously shown in Figure 15 and discussed in Section 3.6.

If you select the option [piwg\_performance\_benchmark [1]] from the control panel in Figure 30, and go to that node and invoke the Navigate menu, then the [Go To Other Occurrence] option will be available from the Navigate menu, and the option [piwg\_performance\_benchmark [2]] will be available from the Other Occurrences submenu. An example of such a cascading menu containing the [Go To Other Occurrence] option is shown in Figure 31.

#### 4.7 Centering a Node

Another method of browsing involves traversing the graph in a topographical manner, *i.e.*, looking at the visible nodes in the view and moving the view itself over the graph by various means. One method of accomplishing this type of browsing is to center a particular node in the view that was previously off to one side of the view. This operation then causes other previously hidden nodes to become visible. This process can be repeated until the desired node has been found. This method of navigation supports a more casual style of browsing than other methods described above. The [Center this Object] or [Center this Category] option is always available from the Navigation menu in Specialization Views, or the [Center this Relation] option in Relationship Views, for this purpose. This style of browsing is similar in principle to using the view's scrollbars to move the graph display.

You can center a node in a view by selecting the [Center this Category], [Center this Object], or [Center this Relation] option in the menu (the appropriate option is available depending on whether the node is a category or an object). This will reposition the graph inside the graph display window so that it is centered, as close as possible, on the selected node.

In the Relationship View there is a unique option available called [Center this Object in Specialization View]. When this option is invoked from the Navigate menu of a node in a Relationship View, the graph display of the Specialization View is centered upon the selected

node. The current focus is placed upon the node in the Specialization Node; however, in this version of the RLF GB the new focus may not be readily apparent because the Specialization View may be occluded by the Relationship View. You may have to use your window manager to raise or lower the appropriate windows to see the effects of this command. Consult the documentation for your window manager for the operational details of raising and lowering windows.

Object: example_quicksort	
Navigate	▷
Perform Action	▷
Advice	Read Description desc_source
Suppress	Extract source
Display Relationships Graphically	View Source source
Display Relationships Textually	

Figure 32: The [Perform Action...] Cascading Menu

#### 4.8 Performing Actions

The categories and objects in an RLF library may contain more than just static information. Any executable program, script, or procedure, called an “action,” may be associated with any category or object by the creator of the library. It is then possible to invoke any of the available actions by selecting the appropriate menu option. This mechanism is typically used to view textual attribute information associated with nodes, but has general applicability to a wide variety of needs.

Actions may be performed by selecting the [Perform Action] option, if it is available. An example of a Perform Action menu is given in Figure 32.

In general, action clauses are displayed in menus with the following format:

*action\_name target\_name*

An action may be invoked on the files associated with the target names “desc\_source” and “source.” In the example menu shown in Figure 32, the description file “desc\_source” may be read, and the source code file “source” may be viewed. If the [View Source source] option is selected, then the RLF GB creates a new window and invokes a text pager, in this case the *view* program (*vi* in read-only mode), on the “source” file. The results of the invoked program are displayed in the new window. See Figure 33.

The type of text pager invoked can be set through the environment variable `RLF_PAGER`. See the appendix for detailed information on setting RLF environment variables.

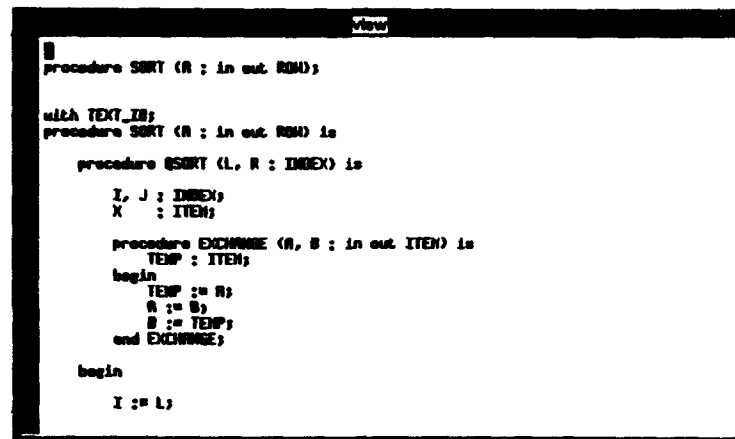


Figure 33: Performing an Action

Category: Sort Algorithms	
Navigate	▷
Perform Action	
Advice	
Suppress	▷
Display Relationships Graphically	
Display Relationships Textually	

Figure 34: A Category Node with the [Advice] Menu Option

#### 4.9 Inferencing—Getting Advice at a Node

In addition to actions, there may be an inferencer attached to the node. The inferencing capability allows you to interact with a rule-driven knowledge base that may be stored in an RLF library. The interaction takes the form of a series of questions that you are asked, and your answers cause the addition or deletion of facts to and from the knowledge base. These facts then allow the inferencer to make intelligent decisions to guide you to an appropriate node in the network. If an inferencer exists at a particular node, then the [Advice] menu option is displayed.

When you select the [Advice] option from the menu, the RLF inferencer is invoked and the dialog appears in a pop-up Advice Dialog Box window. An example of a menu at a node with an attached inferencer is given in Figure 34. See the *RLF Modeler's Manual* for additional information on RLF inferencing capabilities. An example of a pop-up Advice Dialog Box window is given in Figure 35. This figure indicates the components of an Advice Dialog Box: the text entry box, the dialog window, and the command buttons.

To complete the dialog with the inferencer, place the pointer inside the text entry box (located beneath the dialog window; the "caret" cursor becomes an "I-beam" cursor) and type the appropriate option number and then either press the **Return** key or click on the

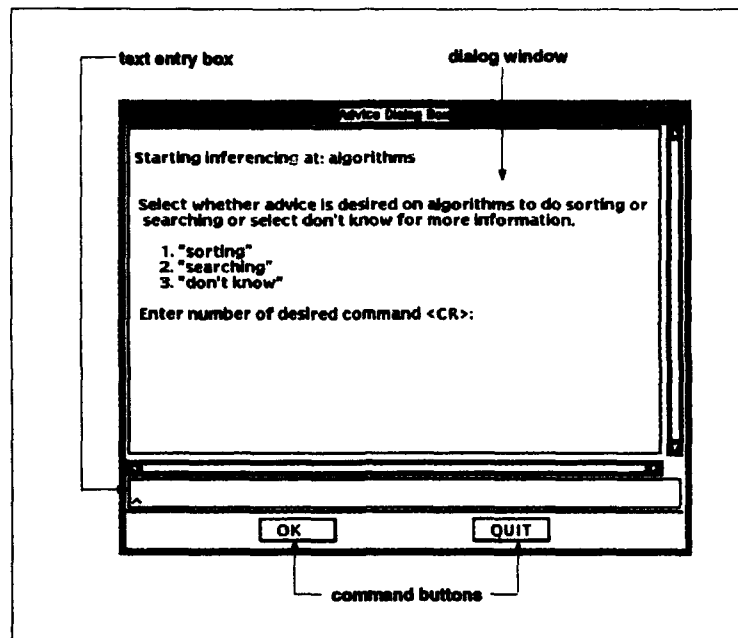


Figure 35: The Components of an Advice Dialog Box

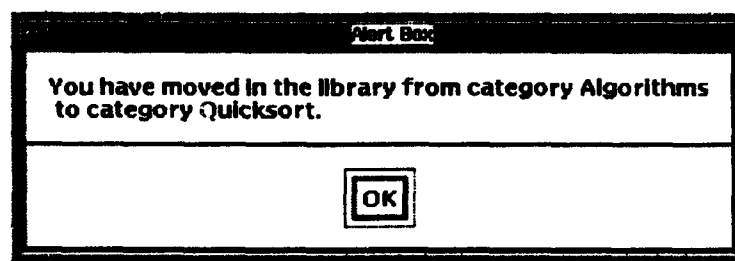


Figure 36: The Alert Box from the [Advice] Menu Option

[OK] command button. After the inferencer finds a node that you are interested in going to for further investigation, and you select the appropriate option, the graph is then traversed to that location and that node is centered in the view. An alert box is then displayed that informs you of the completion of the traversal operation and the node name where you are currently located. An example of this alert box is given in Figure 36.

If the selected category or object does not have an inferencing capability, then the [Advice] menu option will not be available in the menu. If the category or object does have an inferencing capability and you complete a dialog but no suitable node is found, then an alert box is displayed informing you that no more guidance is available for this particular concept. The text of this alert box is as follows:

End of available guidance for this node.

You may also end the inferencing session at any time by clicking on the **[Quit]** command button.

See Appendix D for inferencing session customization options. For example, you may specify the explanation level, or whether moves through the library initiated by the inferencer are automatic or not. If the moves are not automatic, then you are prompted for confirmation.

#### **4.10 Asset Extract and Export Functions**

Underlying all the nodes and network connections, most RLF libraries contain data files that are the reason for having the library and browser in the first place. The library and browser are just applications that make storage and retrieval of the data files into an ordered and convenient process. In previous versions of the RLF GB, two options were available under an Extract menu: **[Extract Files]** and **[Export Asset]**. These functions have been subsumed by the "action" capability in the current version of the browser. It is now the responsibility of the modeler to provide asset extraction and export functions through RLF actions. The action capability was described in a previous section. For a more detailed description of the RLF action capability, see the *RLF Modeler's Manual*.

#### **4.11 Node Suppression**

There may be situations where you would like to reduce, or filter, the amount of information being displayed in the graph display window for any given library. Sometimes large libraries can become unwieldy or difficult to read. Or sometimes for aesthetic reasons you may want to temporarily hide a particular node or subtree for viewing or printing purposes. To deal with these situations, a set of node suppression functions is provided. Each node menu will have either a **[Suppress]** option (if the node has *no* descendants) or a **[Suppress >]** option (if the node has *any* descendants). The following sections describe the operational details of the suppression functions.

##### **4.11.1 Suppressing Categories or Objects with No Descendants**

If a category or object node has no descendants, then the **[Suppress]** option will be available. When you choose the **[Suppress]** option from the menu, the selected category or object and its arc are hidden from view. Once the **[Suppress]** option is selected, the suppression occurs. Suppressed nodes can always be unsuppressed, so don't worry about making a mistake with the suppression commands. In addition to the node-level unsuppress commands, there are view-level unsuppress commands available in the Filter View pull-down menu in the command bar, which were explained in Section 3.4.



#### 4.11.2 Unsuppressing Nodes

By definition, only categories may have descendant nodes. Therefore, only category nodes will have the cascading version (**[Suppress ▸]**) of the Suppress command, indicating a submenu of options is available. Selecting the **[Suppress ▸]** option will cause a cascading menu to be displayed with the following options:

- **[Suppress]**
- **[Unsuppress Children]**
- **[Unsuppress Descendants]**

This **[Suppress]** command works exactly as described in the previous section. The unsuppress commands are what are different about this menu; they provide node-level unsuppression functions. To unsuppress a category or object at the node level, click on a parent or ancestor category of the suppressed category and select either **[Unsuppress Children]** or **[Unsuppress Descendants]** from the menu options. The **[Unsuppress Children]** command will only unsuppress child nodes of the current node, and the **[Unsuppress Descendants]** command will unsuppress all currently suppressed descendants of that node.

There is a subtle but important difference between the definition of a child node and a descendant node. Both children and descendants must fulfill the specialization or individuation relationship; however, a child resides at the *next* lower level of the hierarchy, while a descendant may reside at *any* lower level of the hierarchy.

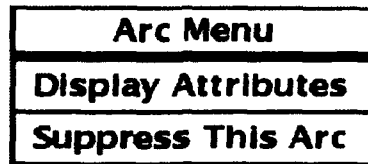


Figure 37: The Arc Menu

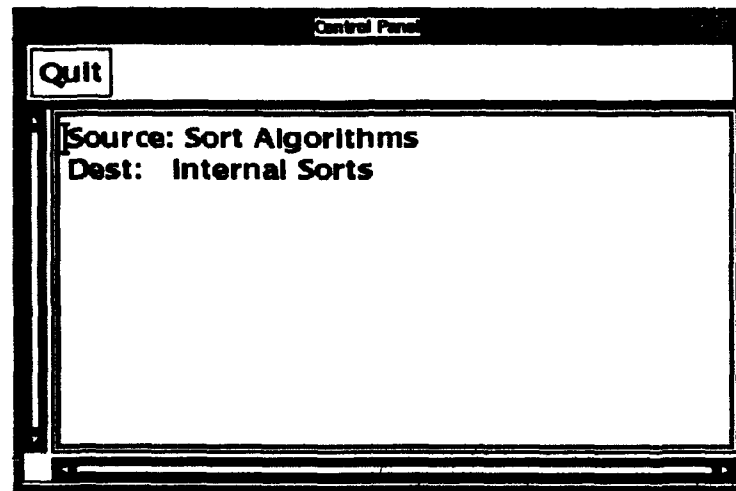


Figure 38: An Arc Attributes Display Window

## 5 Arcs

All arcs are represented with lines and connectors. The connectors are bitmaps, and, like concepts, can be modified to adapt to individual requirements. See Section 3.2, "The Graph Display," and Appendix A, "Customization," for an explanation of how to modify bitmaps. An arc and its connector is one of the view components pointed out in Figure 4.

### 5.1 Displaying the Attributes of an Arc

All arcs contain attributes that can be displayed. To display the attributes of an arc, click on the arc connector. The procedure is the same as for nodes: when you place the pointer over an arc connector, it becomes highlighted; clicking and holding down the mouse button with the pointer on a highlighted connector causes the Arc Menu to be displayed (see Figure 37); and selecting the [Display Attributes] menu option causes the selected arc's attributes to be displayed in a newly created window. An example of an attributes display window is shown in Figure 38.

The arc attributes indicate the source and destination nodes for that arc. For example, the arc attributes shown in Figure 38 indicate that, for that arc, the source node is **Sort Algorithms** and the destination of that particular arc is **Internal Sorts**. This option can be useful in situations where there is a long distance between a source node and

its destination, or in cases where *only* an arc connector is visible. This situation may arise when very large graphs are displayed. You can find out an arc's destination by using the Arc Menu's **[Display Attributes]** option, then you could go to the source or destination node, if desired, by using the Main Menu's **[Go To]** command button to specify the desired node.

## 5.2 Suppressing Arcs from View

When you choose the **[Suppress Arc]** menu option by clicking on it, the selected arc, the node it points to and all of that node's descendants are hidden from view. To unsuppress the arc, click the **[Filter View]** command bar button to pull down its menu, and select the menu option **[Unsuppress All]**; alternatively, click on the source node of the suppressed arc, and select either **[Unsuppress Children]** or **[Unsuppress Descendants]** from the menu options.

## A Appendix: Customization

Since the RLF GB is an X application, the user is able to customize the "look and feel," or appearance and behavior, of the application to a significant degree. The range of customization possible is large, therefore, only the subset of bitmap and font customization options will be discussed in this appendix. For a detailed description of all the possible X customization procedures, see the X references listed in Appendix I, "References."

### A.1 X Resources

All X applications have *resources*. These resources consist of items such as bitmaps, fonts, colors, cursors, and windows. All these resources have unique identifiers associated with them for naming purposes; thus, the user or application programmer can set the values of these resources to customize the look and feel of applications.

There is a set of precedence rules that all X applications follow to determine what resources will be applied to a given invocation of the application. Resources that are loaded first will be overridden by those loaded later.

The following list states the rules for setting X application resources in **reverse** order of priority. For example, item 1 will be overridden by item 4, and item 4 will be overridden by item 5.

1. `/usr/lib/X11/app-defaults/Application_Class_Name`

- the application resource specification file on the host running the client X application
- this corresponds to the `/usr/lib/X11/app-defaults/RLF_Browser` file for the RLF GB application

2. XAPPLRESDIR environment variable

- the value of the XAPPLRESDIR environment variable is a pathname; it can be set to point to a file named *Application\_Class\_Name*, e.g., `RLF_Browser`, that resides somewhere other than `/usr/lib/X11/app-defaults`

3. Resources loaded into the RESOURCE\_MANAGER property of the root window

- typically, the user arranges to have `xrdb` run from the X initialization file `.xinitrc`
- if the RESOURCE\_MANAGER property is not set, the *resource manager* looks for a `.Xdefaults` file in the user's home directory

4. XENVIRONMENT environment variable

- a complete pathname including the filename (different from the XAPPLRES-  
DIR environment variable, which is only the pathname, but does not include the  
filename)
- if this variable is not defined, then the *resource manager* looks for a file in the  
user's home directory named *.Xdefaults-hostname*

#### 5. Command Line Values

- specified with the *-xrm* option on the command line
  - values are loaded for that instance of the program only
6. If the application has defined any command line options by passing an options table to the programmatic X call *XtInitialize*, values from the command line will override those specified by any other resource settings.

A default application resource specification file called *RLF\_Browser* should exist in */usr/lib/X11/app-defaults*. If not, contact your site's system administrator. If nothing else is done, the RLF GB will use the resource values in that file.

The most convenient means of customizing the RLF GB is to make your own copy of the *RLF\_Browser* file and set the environment variable *XAPPLRESDIR* to point to its *directory* location. Then you can edit the *RLF\_Browser* file and change any resources specified in that file, such as bitmaps, fonts, window sizes and window placement to suit individual needs.

## A.2 Bitmaps

The RLF GB, Version 4.0, uses the Motif widget set, therefore. Motif configuration standards and conventions must be used. This has a significant impact on how resources are set, particularly in relation to earlier versions of the RLF GB. The result is that most of the methods for setting resources are different from pre-4.0 versions.

The pathname to the `bitmaps` directory is no longer specified in the `Browser` file. In Motif applications, the `bitmaps` directory must be a subdirectory in the resource file's pathname. For example if you set the `XAPPLRESDIR` environment variable to `/usr/lib/X11/rlf gb`, then the RLF GB looks for the `bitmaps` directory to be at `/usr/lib/X11/rlf gb/bitmaps`. This directory must contain the bitmaps used by the RLF GB.

You can change `XAPPLRESDIR` to point to a different directory, and therefore a different `bitmaps` directory, to use bitmap files that you have created or customized. If the pathname is not specified correctly, the RLF GB will not be able to find its bitmaps, and blank space will be displayed instead. It is important that the resource file's directory pathname be specified correctly and for the `bitmaps` directory to be located in that directory as a subdirectory or else the RLF GB graphic display will be aesthetically less pleasing. The `bitmaps` directory included in the RLF release may be used directly unless use of alternative bitmaps is desired, in which case you may create a `bitmaps` directory in the location of your choice, containing the desired bitmaps.

The specific bitmaps used by the RLF GB to represent displayed objects can be specified in the `RLF_Browser` file. The different types of nodes in an RLF GB graph view each have a bitmap defined for it. For example, all category nodes that do not have any actions or inferencers attached use the bitmap file specified by the following line in the `RLF_Browser` file:

```
vshell*scr.window*node_CATEGORY_KIND.labelPixmap: box_m.xbm
```

while all object nodes that do not have any actions or inferencers attached use the bitmap file specified by the following line in the `RLF_Browser` file:

```
vshell*scr.window*node_OBJECT_KIND.labelPixmap: cube_m.xbm
```

All category nodes could be changed to display a different bitmap by editing the above line in the `RLF_Browser` file and specifying a different value for the bitmap file. For example, to change the category nodes to a bitmap you created and placed in a file called `my_bitmap.xbm`, you would set the value in the `RLF_Browser` resource file to the following:

```
vshell*scr.window*node_CATEGORY_KIND.labelPixmap: my_bitmap.xbm
```

### A.2.1 Bitmap File Naming Convention

This version of the RLF GB allows a finer granularity of bitmap file specification for different node types, as previously described in Section 3.3. This increase in control over the look of the graph display has a disadvantage—the number of bitmap files that must be managed has also increased.

A bitmap file naming convention has been established in an attempt to make the management of a multitude of bitmap files easier. As shown in Figures 7 and 6, there are now eight base types of bitmap files:

- Category
- Category with Action
- Category with Inferencer
- Category with Action and Inferencer
- Object
- Object with Action
- Object with Inferencer
- Object with Action and Inferencer

This fact by itself is not too much of a problem. However, to the above set we must also add a “reverse label pixmap” for each item in the set. The reverse label pixmap is used for highlighting. This brings the total to sixteen basic types. Even sixteen isn’t too bad, but there’s yet another multiplier to deal with—size. Different sizes of bitmaps are necessary because the needs of the user are always changing. Sometimes you may need to give a demonstration, requiring extra large bitmaps, and sometimes you may need to display an extremely large graph, requiring extra small bitmaps so that the application does not exceed the maximum drawable area that the X Window System is able to handle. Therefore a set of the sixteen basic bitmap types is needed for each required size. If we assert that we would like to have five sizes available: extra small (XS), small (S), medium (M), large (L), and extra large (XL), then we have a total of 90 bitmap files needed just to accommodate the node bitmaps.

The bitmap file naming convention has the following structure:

```
{box | cube}[-{A | I}][_rev].{xs | s | m | l | xl}.xbm
```

For example, the following table lists the node type and the corresponding bitmap files for the set of medium-sized bitmaps:

Node Type	Bitmap File
Category	box_m.xbm box_rev_m.xbm
Category with Action	box_A_m.xbm box_A_rev_m.xbm
Category with Inferencer	box_I_m.xbm box_I_rev_m.xbm
Category with Action and Inferencer	box_AI_m.xbm box_AI_rev_m.xbm
Object	cube_m.xbm cube_rev_m.xbm
Object with Action	cube_A_m.xbm cube_A_rev_m.xbm
Object with Inferencer	cube_I_m.xbm cube_I_rev_m.xbm
Object with Action and Inferencer	cube_AI_m.xbm cube_AI_rev_m.xbm

If you wanted to change all the bitmaps from size "medium" to size "extra small," you could perform a global search and replace in the `RLF_Browser` file with your favorite text editor, changing the string `"_m.xbm"` to `"_xs.xbm"`. This would cause all bitmap file specifications to use the extra small bitmaps.



### A.3 Fonts

Fonts used by the RLF GB application can be changed in the same manner as bitmaps. There are lines in the `RLF_Browser` file that specify what fonts to use for particular textual objects in the RLF GB display. For example, the font used for Category concept names in the graph display is specified in the following line in the `RLF_Browser` file:

```
vshell*scr_window*node_label_CATEGORY_KIND*fontList: -b&h-lucida-bold-i-*-17-*****
```

and similarly, this font name could be modified to suit varying needs. For instance, the display of very large graphs sometimes necessitates the use of a smaller font (and/or smaller bitmaps) so that the graph display will not exceed the capacity of X to display it. Or sometimes very large fonts and bitmaps are desired for demonstration purposes. These types of customizations and many others are most easily accomplished by modifying the `RLF_Browser` file and setting the `XAPPLRESDIR` environment variable appropriately.

### A.4 Invoking the Browser from a Shell Script

The C shell script `RLF_GB` checks the appropriate environment variables to determine as best it can whether they are valid, and then invokes the RLF Graphical Browser. This method of invocation is recommended for novice and beginning RLF users. The `RLF_GB` script is found in the `bin` directory of this software release, along with the `Graphical_Browser` executable.

### A.5 Command Line Arguments for Setting X Resources

Another way of customizing the look of the RLF GB is to set X resources via the command line. Various X resources can be specified by invoking the X application with the appropriate command line arguments. As can be seen in the list of precedence rules, this method will override any other previous settings.

An example of invoking the RLF GB with command line arguments is as follows:

For an extra-large library, use a small font and small bitmaps:

```
Graphical_Browser \
-xrm "vshell*scr_window*node_label_CATEGORY_KIND*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ACTION*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ADVICE*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ADVICE_AND_ACTION*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_OBJECT_KIND*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ACTION*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ADVICE*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ADVICE_AND_ACTION*fontList: -b&h-*-r-*-10-*****" \
-xrm "vshell*scr_window*node_CATEGORY_KIND.labelPixmap: box_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_KIND.reverseLabelPixmap: box_rev_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ACTION.labelPixmap: box_A_xs.xbm" \
```

```

-xrm "vshell*scr_window*node_CATEGORY_W_ACTION.reverseLabelPixmap: box_A_rev_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE.labelPixmap: box_I_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE.reverseLabelPixmap: box_I_rev_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE_AND_ACTION.labelPixmap: box_AI_xs.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE_AND_ACTION.reverseLabelPixmap: box_AI_rev_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_KIND.labelPixmap: cube_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_KIND.reverseLabelPixmap: cube_rev_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ACTION.labelPixmap: cube_A_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ACTION.reverseLabelPixmap: cube_A_rev_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE.labelPixmap: cube_I_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE.reverseLabelPixmap: cube_I_rev_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE_AND_ACTION.labelPixmap: cube_AI_xs.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE_AND_ACTION.reverseLabelPixmap: cube_AI_rev_xs.xbm"

```

For a demonstration, you might want to use a large font and large bitmaps:

```

Graphical_Browser \
-xrm "vshell*scr_window*node_label_CATEGORY_KIND*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ACTION*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ADVICE*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_CATEGORY_W_ADVICE_AND_ACTION*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_OBJECT_KIND*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ACTION*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ADVICE*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_label_OBJECT_W_ADVICE_AND_ACTION*fontList: -b&h-*-r-*-20-*-*-*-*-*" \
-xrm "vshell*scr_window*node_CATEGORY_KIND.labelPixmap: box_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_KIND.reverseLabelPixmap: box_rev_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ACTION.labelPixmap: box_A_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ACTION.reverseLabelPixmap: box_A_rev_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE.labelPixmap: box_I_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE.reverseLabelPixmap: box_I_rev_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE_AND_ACTION.labelPixmap: box_AI_m.xbm" \
-xrm "vshell*scr_window*node_CATEGORY_W_ADVICE_AND_ACTION.reverseLabelPixmap: box_AI_rev_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_KIND.labelPixmap: cube_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_KIND.reverseLabelPixmap: cube_rev_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ACTION.labelPixmap: cube_A_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ACTION.reverseLabelPixmap: cube_A_rev_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE.labelPixmap: cube_I_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE.reverseLabelPixmap: cube_I_rev_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE_AND_ACTION.labelPixmap: cube_AI_m.xbm" \
-xrm "vshell*scr_window*node_OBJECT_W_ADVICE_AND_ACTION.reverseLabelPixmap: cube_AI_rev_m.xbm"

```

The above command-line invocations of the RLF GB with command line arguments specify the font and bitmaps of the graph view window that is displayed. These command line arguments override any other resource settings that may have been specified previously. Editing startup scripts like these, or typing new command line arguments manually, are more examples of the many options available for customizing X applications such as the RLF GB. Other X resource customizations, such as window size and placement, can also be accomplished using the command line argument method.

For further possibilities of customizing X resources, consult the X Window System reference given in Appendix I, References.

## B Appendix: Environment Variables

With the UNIX operating system, there is some special information that is a part of your user environment. A set of special variables, called *environment variables*, maintains this data. UNIX can pass the values of these variables to programs executed from the shell. Unlike C shell variables that are accessible only from within the shell, environment variables are available to both your current shell and subsequent programs.

When you execute a program such as the *Graphical Browser*, UNIX gives it the values of all the environment variables. You can define environment variables (or change their values) from a shell like the C shell. The notation is as follows:

```
setenv VARNAME string
```

The above command would define **VARNAME** to be an environment variable and initialize its value to **string**. By convention, the variable name is in uppercase.

The RLF GB checks to see whether certain environment variables are set or not. If any of the predefined environment variables are set, then the RLF GB will use their values. If no environment variables are set, then it will use its own predefined default values.

The environment variables that the RLF GB looks for, and their default values, are as follows:

- **RLF\_PAGER** *less*
- **RLF\_EDITOR** *vi*

The **RLF\_PAGER** environment variable defines what program is used for text paging actions when they are invoked from the RLF GB. The *less* program is a public domain program that is similar to the UNIX *more* program, but which allows backward movement in the file as well as forward movement. Also, *less* does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like *vi*.

The **RLF\_EDITOR** environment variable defines what program is used for text editing actions when they are invoked from the RLF GB. The *vi* program is the standard UNIX text editor.

To change an environment variable, use the **setenv**, as in the following example:

```
% setenv RLF_EDITOR emacs
```

The above example would cause the RLF GB to invoke the public-domain text editor program *emacs* instead of *vi* whenever a text-editing action was called for from the RLF GB.

## C Appendix: Command Line Options

Command line options allow yet another way to tailor the behavior of the RLF GB. Setting command line options allows you to change the behavior of each invocation of the RLF GB, whereas environment variables are generally used on a per-login or site-specific basis.

Command line options are issued from the shell when the RLF GB is invoked. In general, command line options have the following format:

```
% Graphical_Browser -option1 value1 -option2 value2 ...
```

You may get a list of all the available command line options by typing the following:

```
% Graphical_Browser -help
```

The above command will produce the following output:

```
Welcome to the RLF Graphical Browser.  
Version 4.0
```

```
Copyright 1992, Paramax Systems Corp.
```

Available command line arguments:

-help	prints available command line arguments
-I <pname>	uses RLF libraries found in the directory specified by pathname <pname>
-l <name>	specifies name of the library to browse
-e <fname>	specifies filename of editor to use to edit text associated to a network
-p <fname>	specifies filename of pager used to view text associated to a network
-d	enables debug messages from the RLF tools

Each one of the options in the above list is explained in the following paragraphs.

The -I option changes the pathname where RLF libraries are searched for. This option can be used to override any existing RLF\_LIBRARIES environment variable.

The `-l` option can be used to specify the name of a particular library to be browsed when the RLF GB is invoked. If the library name contains spaces, then the name must be enclosed in quotes, as in the following example:

```
% Graphical_Browser -l "Sort and Search"
```

The `-e` option can be used to specify which text editor is used for text editing actions when they are invoked from the RLF GB. This option can be used to override the `RLF_EDITOR` environment variable described in the previous section.

The `-p` option can be used to specify which text pager is used for text paging actions when they are invoked from the RLF GB. This option can be used to override the `RLF_PAGER` environment variable described in the previous section.

The `-d` option can be used to produce large quantities of diagnostic output from the RLF and the RLF GB. You may or may not find this option useful. This option is mostly intended to be used by developers and support personnel if the need to locate the source of a software defect occurs.

## D Appendix: The RLF Initialization File

The RLF checks for an initialization file upon startup. The initialization file is looked for in your home directory. The name of the initialization file is:

`.rlfrc`

All the command-line options described in Appendix C can alternatively be set through the initialization file mechanism.

The format of the `.rlfrc` file in BNF format is as follows is given in a following section.

A sample `.rlfrc` file is given below. This example is similar to the `.rlfrc` file that is included with the RLF release. The data types are described following the example.

```
-- Where to find the library instances
library directory: /libraries/instances

-- Default library
library: "Sort and Search"
-- Start at the root category
initial category: Thing

-- View settings
topology: on
cardinality: on
layout offset: x: 5
layout offset: y: 7
view depth: 3
view type: specialization

-- Bitmaps
node bitmap: category: actions: /bitmaps/cat_actions.xbm
node bitmap: category: inferencer: /bitmaps/cat_inf.xbm
node bitmap: category: attributes: /bitmaps/cat_attrs.xbm
node bitmap: object: /bitmaps/object.xbm

-- Library advice settings
advice: explanations: reasoning moving
advice: automatic move: false

-- Specification translator settings
translator: Lmdl: quiet: yes
```

```
translator: Lmdl: only: model
translator: Rbdl: quiet: no
```

### D.0.1 Library Instances

You can specify a default pathname for the location of the RLF libraries. This pathname must be an absolute pathname. When the RLF is invoked, it will use this pathname as the location of the RLF libraries to be used as input.

```
-- Where to find the library instances
library directory: absolute_pathname
```

### D.0.2 Default Library

A default RLF library may be specified. The specified library will be displayed by the RLF GB automatically upon invocation; the Main Menu will not be displayed. If the specified library contains spaces, then the name should be enclosed in double-quotes, as in the above example.

```
-- Default library
library: RLF_Library_Name
```

### D.0.3 Initial Category

You can also specify a concept that will become the node of current focus. The default is to have the root node become the node of current focus; this specification changes that default.

```
-- Start at the root category
initial category: concept_name
```

### D.0.4 View Setting

There are various view settings that can also be tailored through the *.rlfrc* file to suit your needs. Most of these settings are self-explanatory from their names. The *topology* setting allows you to specify whether the Topology Display will be displayed or not upon startup. The *cardinality* setting affects the display of the number restrictions of relationships in Relationship Views (e.g., (0..infinity) or (1..1)). You may or may not find the cardinality information useful. If you do not find it useful, you can prevent it from being displayed

by specifying this setting as *off*. The *x* and *y* coordinate offsets (how many pixels in the horizontal and vertical dimensions, respectively) can be set through the *layout offset* setting. This setting is typically used to compress the white space out of a graph display. The view depth setting affects how many levels of the graph are initially displayed. The default is to display all levels; this setting changes that default. Finally, the initial view type may be specified. The default is to display the Specialization View, but you may change this to be the Relationship View.

```
-- View settings
topology:  on or off
cardinality:  on or off
layout offset:  x:  positive_integer
layout offset:  y:  positive_integer
view depth:  positive_integer
view type:  specialization or relationship
```

### D.0.5 Bitmaps

You can also change bitmap settings through the *.rlfrc* file. A pathname to the location of a different set of bitmap files may be specified.

Note that the most general form sets the bitmap for all variations. For example, "node bitmap : category : box.xbm" will set the bitmap for all kinds of categories, and would override "node bitmap : category : actions : box\_A.xbm". This is implemented this way so that if you want to set just the bitmap for categories or objects, you do not need to iterate over all variations. Also, if you forget a variation, the default covers the omission.

```
-- Bitmaps
node bitmap:  category or object:  actions:  pathname
node bitmap:  category or object:  inferencer:  pathname
node bitmap:  category or object:  attributes:  pathname
```

### D.0.6 Library Advice Settings

Library advice settings affect the behavior of the AdaTAU inferencer. The explanation level may be set here; if it is not set here, then you are prompted for an explanation level when an inferencing session is started. You may also specify whether moves through the library initiated by the inferencer are automatic or not. If the moves are not automatic, then you are prompted for confirmation.



**-- Library advice settings**advice: explanations: reasoning *or* moving *or* questionsadvice: automatic move: true *or* false**D.0.7 Specification Translator Settings**

This category of *rlfrc* settings does not affect the RLF GB; they affect the language translators. See the *RLF Modeler's Manual* for further details.

**-- Specification translator settings**translator: Lmdl: quiet: yes *or* notranslator: Lmdl: only: model *or* statetranslator: Rbdl: quiet: yes *or* no**D.0.8 The RLF Initialization File BNF**

The complete BNF for the RLF initialization file, *.rlfrc*, is provided here as a reference.

```
startup_file =
    {setting}
```

```
setting =
    default_directory |
    default_library |
    start_category |
    view_type |
    view_depth |
    topology_flag |
    cardinality_flag |
    layout_offset |
    bitmap |
    tau_setting |
    debug_flag |
    working_directory |
    history_list_length |
    default_editor |
    default_pager |
    translator_setting |
    comment
```

```
default_directory =
    'library' 'directory' ':' pathname
```

```
default_library =  
    'library' ':' name  
  
start_category =  
    'initial' 'category' ':' name  
  
view_type =  
    'view' 'type' ':' agg_or_spec  
  
agg_or_spec =  
    'relationship' | 'specialization'  
  
view_depth =  
    'view' 'depth' ':' [agg_or_spec ':' ] depth_setting  
  
depth_setting =  
    'all' | integer  
  
topology_flag =  
    'topology' ':' flag_setting  
  
flag_setting =  
    'yes' | 'no' | 'true' | 'false' | 'on' | 'off'  
  
cardinality_flag =  
    'cardinality' ':' flag_setting  
  
layout_offset =  
    'layout' 'offset' ':' [x_or_y ':' ] integer  
  
x_or_y =  
    'x' | 'y'  
  
bitmap =  
    'node' 'bitmap' ':' category_or_object  
    [ ':' has_attribute {has_attribute} ] ':' pathname  
  
category_or_object =  
    'category' | 'object'  
  
has_attribute =  
    'inferencer' | 'actions' | 'attributes'  
  
tau_setting =
```

```
    'advice' ':' tau_setting_type

tau_setting_type =
    'explanations' ':' explanation_type |
    'automatic' 'move' ':' flag_setting

explanation_type =
    'none' | 'all' | explanation_kind {explanation_kind}

explanation_kind =
    'reasoning' | 'questions' | 'moving'

debug_flag =
    'debug' ':' flag_setting

working_directory =
    'working' 'directory' ':' pathname

history_list_length =
    'history' 'length' ':' integer

default_editor =
    'editor' ':' pathname

default_pager =
    'pager' ':' pathname

translator_setting =
    'translator' ':' translator_type

translator_type =
    'lmdl' ':' lmdl_setting |
    'rbd1' ':' rbd1_setting

lmdl_setting =
    quiet_translation | translate_only | default_input_spec

rbd1_setting =
    quiet_translation | default_input_spec

quiet_translation =
    'quiet' ':' flag_setting

translate_only =
    'only' ':' model_or_state
```

```
model_or_state =  
    'model' | 'state'  
  
default_input_spec =  
    'default' 'specification' ':' pathname  
  
comment = {whitespace} '--' {character}  
  
integer = digit {digit}  
  
name = identifier | string  
  
identifier = letter {[underline] letter_or_digit}  
  
letter_or_digit = letter | digit  
  
letter = upper_case_letter | lower_case_letter  
  
pathname =  
    not_colon_double_quote_whitespace  
    {not_colon_double_quote_whitespace}
```

## E Appendix: Error Messages

The informational type of error messages you may encounter while using the RLF GB have already been described in the preceding sections of this manual. This section describes other error messages that may be encountered when running the RLF GB. These messages are usually of a more serious nature and indicate a serious obstacle has been encountered that inhibits further processing.

### E.1 X Window System Errors

Since the RLF GB is an X application, it relies on X to manage its windowing, graphics display, and event handling capabilities. Many different kinds of problems can arise due to this fact, and their severity ranges from insignificant to fatal. Most of these problems are outside the scope of this manual, but some may be due to trivial causes.

#### Error Message:

```
** MAIN PROGRAM ABANDONED -- EXCEPTION "constraint_error" RAISED
```

**Explanation:** This error message usually indicates some type of problem with the X Window System. This error message appears immediately when the DISPLAY environment variable is not set or is set incorrectly. There are two basic forms of the DISPLAY environment variable:

*hostname:0.0*

or

*Internet\_address:0.0*

For example, either of the following ways would be correct for setting the DISPLAY environment variable:

```
setenv DISPLAY owl:0.0
```

or

```
setenv DISPLAY 128.127.161.18:0.0
```

where "owl" is a host name, and "128.127.161.18" is an Internet address. The DISPLAY environment variable tells X applications which X server to connect to, unless it is overridden by the "-display" command line option. The ":0.0" refers to the server and screen number, and only the ":0" is required in most cases.

The `DISPLAY` variable should be set to the Internet address when you are running across a Sun-3 to Sun-4 link, *e.g.*, you're actually running the RLF GB on a Sun-4 host but are viewing the results in a window on a Sun-3 host, or vice versa. The above error may occur in this situation if the `DISPLAY` environment variable is not set to the Internet address form.

The above error message may also occur when the RLF GB is invoked with inappropriate resource specifications. Particularly, if you invoke the RLF GB, receive the initial textual status statements, and *then* receive the above error message, then the following actions are recommended.

Ensure that the `XAPPLRESDIR` environment variable is set properly—it must point to a directory that contains a readable `RLF_Browser` file. Additionally, if you increase font sizes, the RLF GB may have to be invoked with command line options to ensure that the correct window geometries are specified for the windows, or the appropriate geometry specifications updated in the `RLF_Browser` file.

**Error Message:**

Unexpected exit from browser. Unhandled event: <event\_type>

**Explanation:** This message may appear occasionally and can usually be ignored. It usually means that somehow the communication between the X server and the X application was momentarily disrupted. If it occurs frequently or constantly, then it's an indication of a more serious problem, such as incompatible versions of the X server and X application, or improper X installation at your site.

**Error Message:**

X error: failed request ...  
Bad value: integer value out of range ...

**Explanation:** X error messages of this type, or similar X error messages, indicate an incompatibility between the X application (the RLF GB) and your window system. It is recommended that you use the unmodified X Window System, Release 4 or 5, from the X Consortium (also known as "MIT X").

## E.2 OpenWindows Resources

If you use OpenLook from Sun and the OpenLook Window Manager (*olwm*), you may experience some incompatibilities. To minimize incompatibilities with OpenLook, place the following resource specifications in your `.Xdefaults` file in your home directory:

March 1993

STARS-UC-05156/013/00

```
OpenWindows.SetInput: followmouse  
OpenWindows.FocusLenience: true  
*input: true
```

and re-initialize the X server resource data base.

### E.3 RLF Errors

Obviously, the RLF GB relies on the RLF for its knowledge base operations. Since RLF is also a large and complex system, many problems may be encountered that are due to usage of the RLF. Most of the RLF type of errors should have been eliminated by the time the RLF GB is used for browsing any particular RLF library. For example, most errors will probably occur during the library construction phase and RLF will have detected them and reported on them prior to your being able to browse that library. However, it is still not impossible for some RLF processing anomalies to occur.

#### Error Message:

```
** MAIN PROGRAM ABANDONED -- EXCEPTION "FILE_NAME_ERROR" RAISED
```

or

```
** MAIN PROGRAM ABANDONED -- EXCEPTION "NAME_ERROR" RAISED
```

or

```
** MAIN PROGRAM ABANDONED -- EXCEPTION "END_ERROR" RAISED
```

**Explanation:** Your UNIX environment variable `$RLF_LIBRARIES` has either not been set, or has been set incorrectly. To determine the current value of the variable, type:

```
% echo $RLF_LIBRARIES
```

If the variable is not defined, you will receive an error message similar to:

```
RLF_LIBRARIES: Undefined variable.
```

If the `$RLF_LIBRARIES` environment variable has been defined, you should receive a path-name for a directory. To determine if this is a valid RLF library directory, type:

```
% ls $RLF_LIBRARIES
```

If this is a valid RLF library, the results of the `.ls` command should look similar to the following excerpt:



ALL	NETAAACABROLE	NETCAACABGC
AdaNET_States	NETAAACABSATIS_TBL	NETCAACABGEN_OWN_TBL
CBABAB.HYB	NETAAACABSPEC_TBL	NETCAACABINDIV
	:	
	:	
KNETDAABABSUBROLE	NETBAACABIND_OWN_TBL	NETDAACABSUBROLE_TBL
NETAAACABGC	NETBAACABRNG_RESTR_TBL	NETDAACABVAL_RESTR_TBL
NETAAACABGEN_OWN_TBL	NETBAACABROLE	STATE
NETAAACABINDIV	NETBAACABSATIS_TBL	Taustuff/
NETAAACABINDIV_TBL	NETBAACABSPEC_TBL	Text/
NETAAACABIND_OWN_TBL	NETBAACABSUBROLE_TBL	UID_FILE
NETAAACABRNG_RESTR_TBL	NETBAACABVAL_RESTR_TBL	

**Error Message:**

**\*\* MAIN PROGRAM ABANDONED -- EXCEPTION "UNINITIALIZED\_UID" RAISED**

**Explanation:** This message indicates that the RLF library may have been left in a "locked" state. Check if any files exist in the RLF\_LIBRARIES directory with a file extension of ".LOK". If any \$RLF\_LIBRARIES/\*.LOK files do exist, then the "lock" files must be deleted before the library can be browsed again. This may happen if the RLF GB, or some other RLF application, such as the SNDL translator, are terminated abruptly and the RLF libraries therefore not gracefully closed.

**Error Message:**

**\*\* MAIN PROGRAM ABANDONED -- EXCEPTION "UNINITIALIZED\_OBJECT" RAISED**

**Explanation:** This message indicates that the RLF library may have become corrupted due to a system crash or some other type of system failure. Usually the only solution to this problem is to re-translate the offending RLF library.

**Error Message:**

**RLF error in displaying concept attributes.**

**Explanation:** This message indicates that an internal RLF error has occurred while attempting to obtain the attributes of a concept. The knowledge base will have to be examined. The first method of examination could be to use one of the textual browsers to

examine that particular concept. Beyond that, the only recourse is to examine the network specifications for that particular RLF library.

**Error Message:**

Unknown error detected in RLF GB. Continuing...

**Explanation:** This error message is received when an error occurs for which no error handling has been established. If you receive this error message, it is likely that the only way to determine the cause of the error will be to put the RLF GB into a debugger and attempt to repeat the sequence of operations that caused the error.

**E.4 File Processing Errors**

These error messages pertain to file processing. The only file processing done by the RLF GB is when it attempts to read its knowledge base through calls to the core RLF routines. The persistent form of the knowledge base is stored as files. Errors in file processing of this type will be manifested as RLF error messages. These error messages were described in the previous section, "RLF Errors."

In this version of the RLF GB, all external file processing is accomplished through the use of the RLF "actions" mechanism. This means that if any external files are to be viewed or manipulated in any way, then the method to perform that task will have been prescribed by the library modeler. For example, the modeler may have specified any of several possible procedures to view the contents of a file: *vi* in read-only mode, the *more* command (the UNIX text pager program), the *less* command (a public-domain text pager program), or others. If the argument to these commands is erroneous, then file processing errors will result. For example, if the argument specifies a file that is nonexistent or inaccessible to the user due to incorrect permission settings, then the command will probably fail and produce some error message. If any of these types of error messages are received, it is usually beyond the capacity of the RLF GB to fix the problem by itself. Usually the problem is due to external file system factors and will need to be fixed by you, another user, or the system administrator.

Files stored in an RLF library are found in the directory:

- `$RLF_LIBRARIES/Text`

or

- `$RLF_LIBRARIES/Text/modelName`

where *modelName* is the name of an RLF library. (It is recommended that you use the convention of using the RLF library's name as the subdirectory name in the `$RLF_LIBRARIES/Text` directory. Following a convention of this sort will help you to keep the files you are storing more maintainable.) If there are file processing errors encountered while using the RLF GB, then the above directories are the first place to look when checking the status of the affected files.

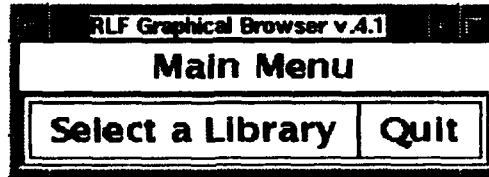


Figure 39: The Main Menu

## **F Appendix: Detailed Usage Scenarios**

This appendix provides a detailed usage scenario. The intent of this section is to show an example of how the RLF GB might typically be used, although, in the interest of brevity, less functions are shown in this example than may be used by a typical browser in a typical browsing session. A truly typical usage scenario would probably consist of some superset of these operations.

### **F.1 Scenario 1: The Candidate Component is Known—Browsing**

In this scenario, the user needs a particular reusable component—in this case a sorting routine. In this scenario, the user knows what kind of sorting routine is the most suitable for the application—a Quicksort routine.

#### **STEP 1.**

Invoke the RLF Graphical Browser from the command line:

```
% Graphical_Browser
```

The Main Menu appears, as shown in Figure 39. Display the menu of libraries by clicking the [Select a Library] button.

#### **STEP 2.**

Select the desired library. In this case, select the "Sort and Search" library, as shown in Figure 40.

#### **STEP 3.**

The graph display view of the "Sort and Search" appears, as shown in Figure 41. The user clicks on the root node to invoke its node menu. (The action of clicking on the node is represented by the arrow-pointer with a dashed line leading to the node.)

#### **STEP 4.**

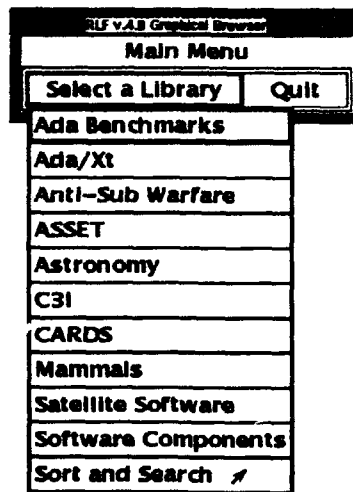


Figure 40: A Library Menu

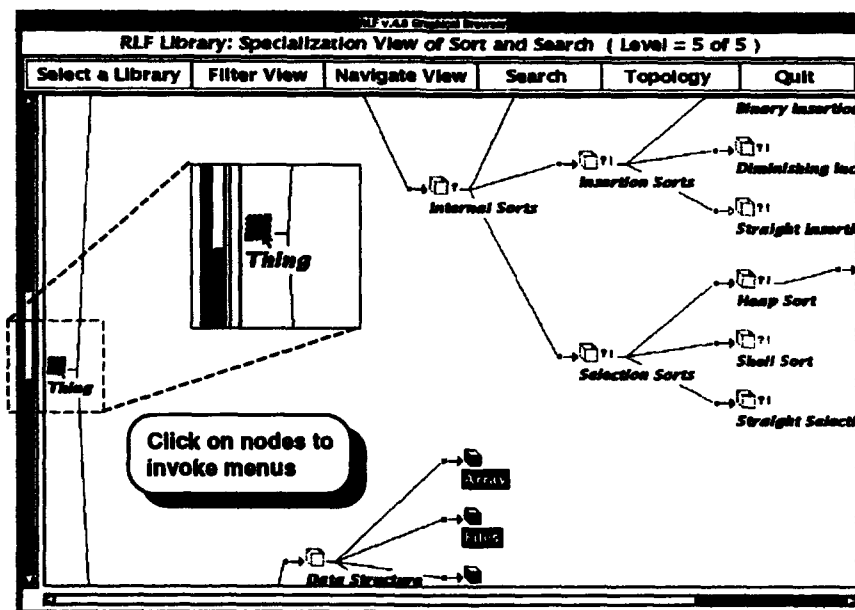


Figure 41: A Graph Display View

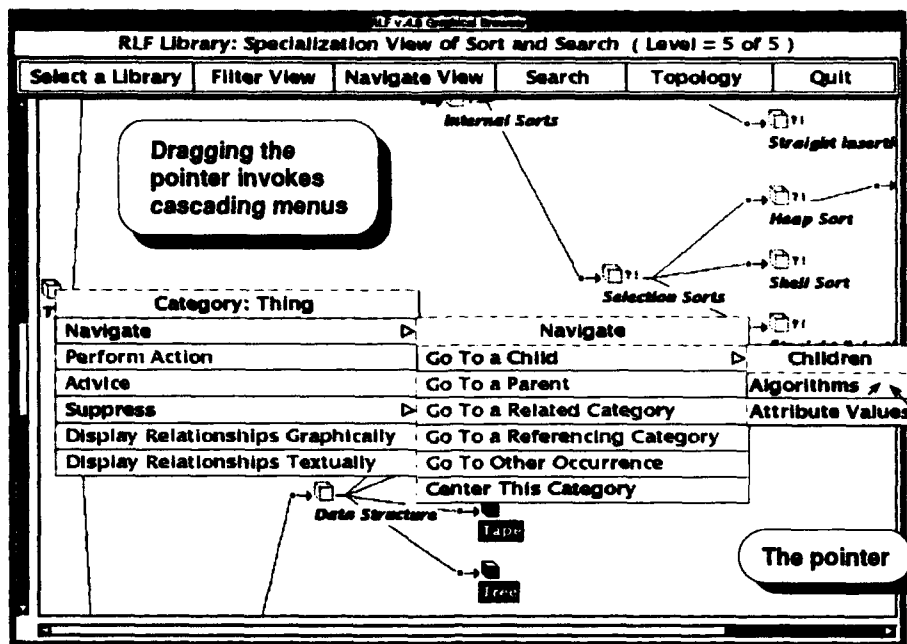


Figure 42: A Category Node Menu

The category node menu appears, as shown in Figure 42. The user clicks on the [Navigate >] option to invoke the Navigate menu.

#### STEP 5.

The Navigate menu is displayed, as shown in Figure 42. The user clicks on the [Go To a Child] option to obtain a list of the root node concept's specializations, or child nodes.

#### STEP 6.

The list of child nodes is displayed in the Children menu, as shown in Figure 42. The user knows that a type of algorithm is needed, so the option "Algorithms" is clicked on.

#### STEP 7.

This process of walking down the specialization hierarchy is repeated by clicking on the nodes "Sort Algorithms," "Internal Sorts," "Exchange Sorts," "Quicksort," and invoking their Go To Child menus, as shown by the arrow pointers in Figure 43. Finally the object node "example\_quicksort" is reached.

#### STEP 8.

After reaching the "example\_quicksort" node, the user wants to look at the source code file associated with that node. The user clicks on the "example\_quicksort" object node and invokes its object node menu, and then clicks on the [Perform Action >] option, as shown in Figure 46.

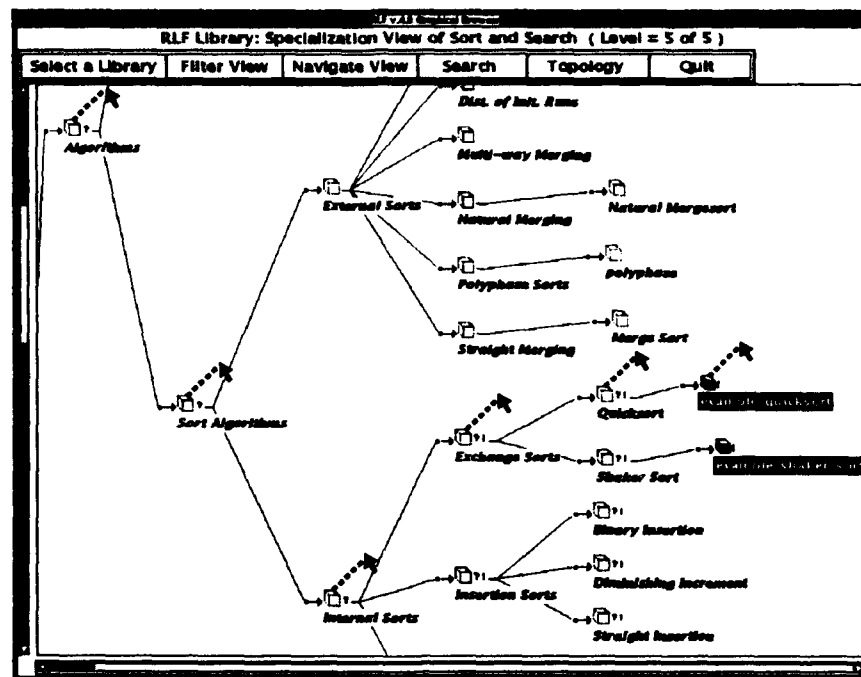


Figure 43: Walking the Specialization Hierarchy

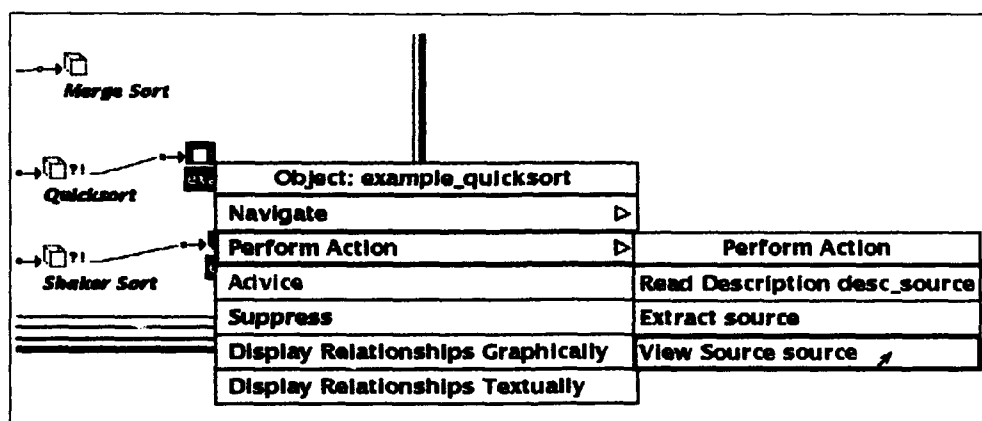
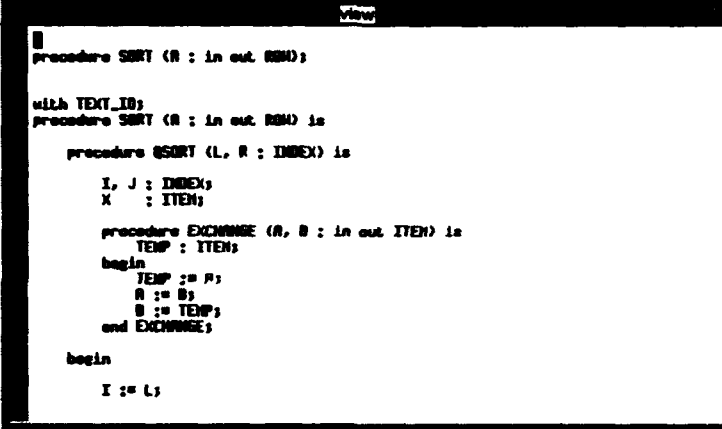


Figure 44: An Object Node Menu



```

procedure SORT (A : in out RM);
with TEXT_IO;
procedure SORT (A : in out RM) is
  procedure QSORT (L, R : INDEX) is
    I, J : INDEX;
    X : ITEM;
    procedure EXCHANGE (A, B : in out ITEM) is
      TEMP : ITEM;
    begin
      TEMP := A;
      A := B;
      B := TEMP;
    end EXCHANGE;
  begin
    I := L;
  end QSORT;
begin
  SORT (A);
end SORT;

```

Figure 45: Reusable Component Source Displayed in a Text Pager

### STEP 9.

The actions provided for the "example\_quicksort" node are displayed in a menu, and the user decides to view the source code by clicking on the [View Source source] option, as shown in Figure 44.

### STEP 10.

A text pager is invoked that allows the user to peruse the source code, as shown in Figure 45. The user pages through the text using the commands of the text pager. When finished with viewing the text, the user issues the appropriate exit command. (In the case of the *view* program, exit by typing either "ZZ" or ":q".

### STEP 11.

The user decides that this is the component that is needed, so the Perform Action menu is again invoked from the object node menu. This time, the user clicks on the "Extract source" option of the Perform Action menu, as shown in Figure 46, and the source code is copied into the user's current working directory.



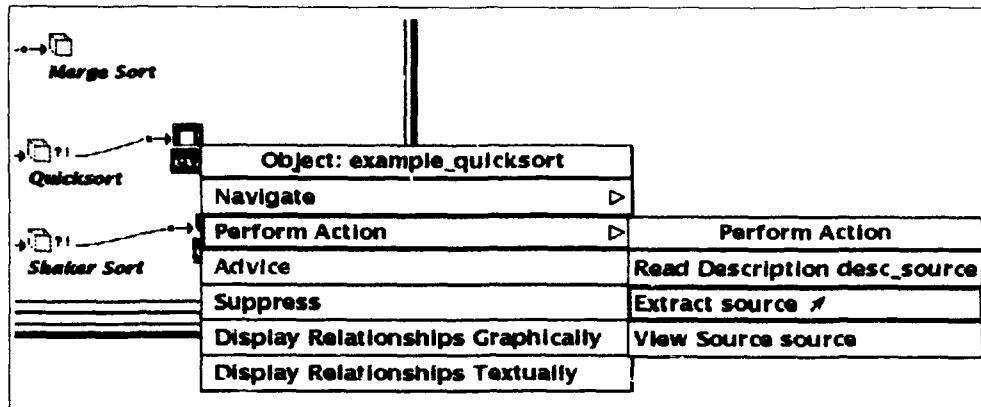


Figure 46: Selecting the [Extract source] Option from a Perform Action Menu

## F.2 Scenario 2: The Candidate Component is Unknown—Guided Search

In this scenario, the user needs a particular reusable component—a sorting routine—but in this scenario, the user does not know what kind of sorting routine is the most suitable for the application. The only knowledge the user has to conduct the component search with is that the candidate component is some kind of sorting algorithm.

### STEP 1.

:

### STEP 6.

Repeat Steps 1 through 6 as in the previous section.

### STEP 7.

In this scenario, the user focuses upon the “Algorithms” concept and invokes the node menu, but instead of selecting a [Navigate] function, the user elects to invoke the [Advice] option, as shown in Figure 47 since the exact kind of algorithm needed is unclear.

### STEP 8.

Selecting the [Advice] option causes a dialog session to be initiated with the user. Questions are asked of the user to help determine the user’s needs. The answers that the user provides help the RLF GB make inferences as to what area of the library may be of the most benefit to the user. The RLF GB then guides the user to that area of the library.

The dialog session occurs in a pop-up Advice Dialog Box window. See Figure 48 for an example of an initial Advice Dialog Box.

Figures 49 through 52 show examples of a dialog session that might occur; the numbers that appear before the “caret” prompts are what the user types in—they indicate what menu option will be selected from the given menu. The user presses the **Return** key or clicks on

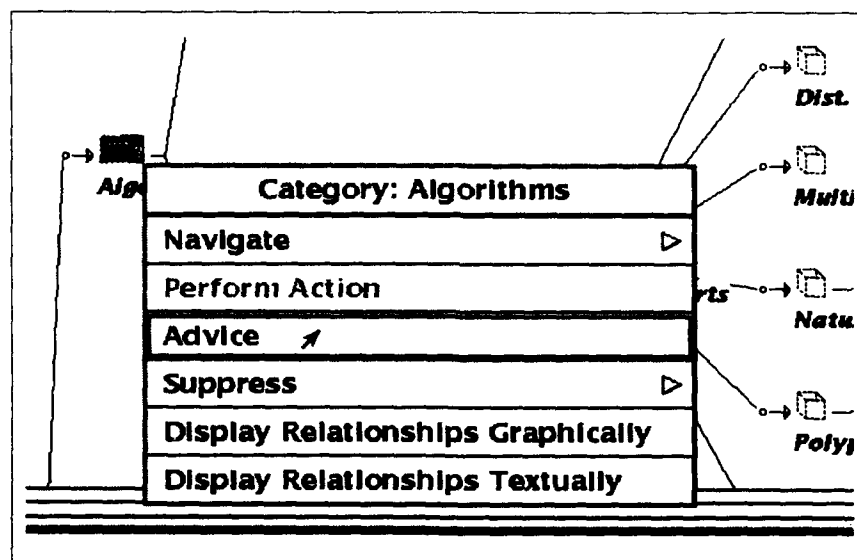


Figure 47: The "Algorithms" Menu—Getting Advice

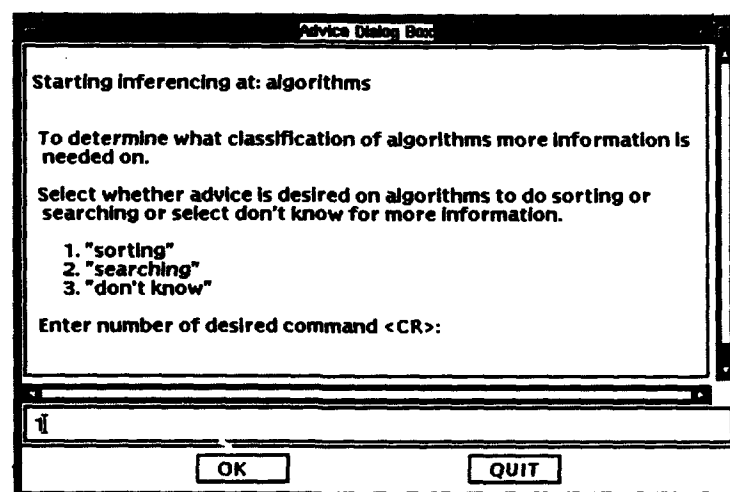


Figure 48: An Example Initial Advice Dialog Box

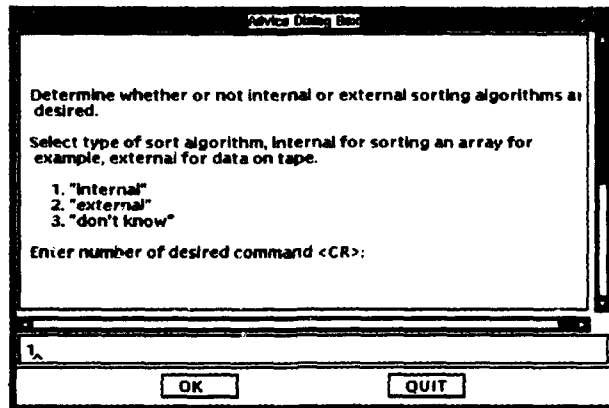


Figure 49: A Second Example Advice Dialog Box

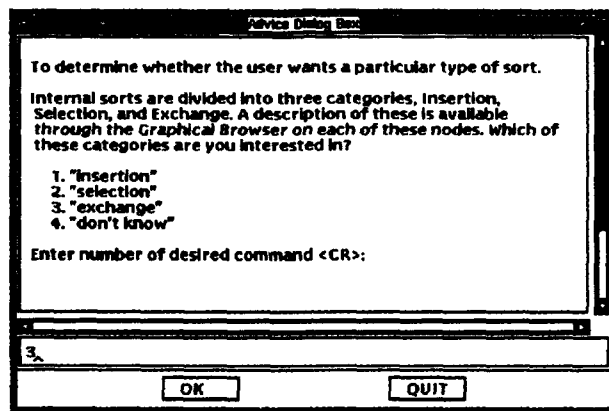


Figure 50: A Third Example Advice Dialog Box

the [OK] command button to enter the selection.

#### STEP 9.

The RLF GB has ended its inferencing session dialog and the user is now focused upon the "Quicksort" category. An alert box is displayed that informs the user of the current location in the library, as shown in Figure 53. The current focus is placed upon the "Quicksort" category node.

At this point, as in the previous scenario, the user would proceed to examine and possibly extract the contents of the "example\_quicksort" component.

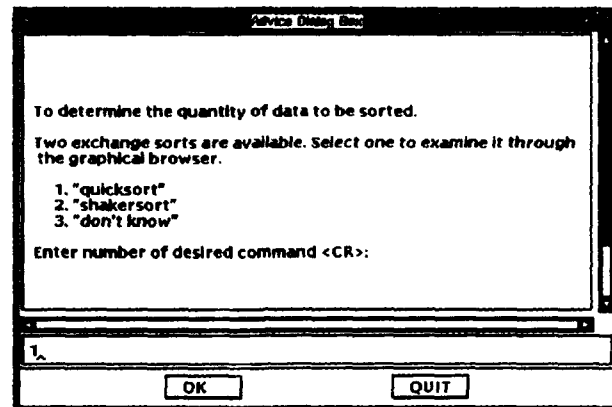


Figure 51: A Fourth Example Advice Dialog Box

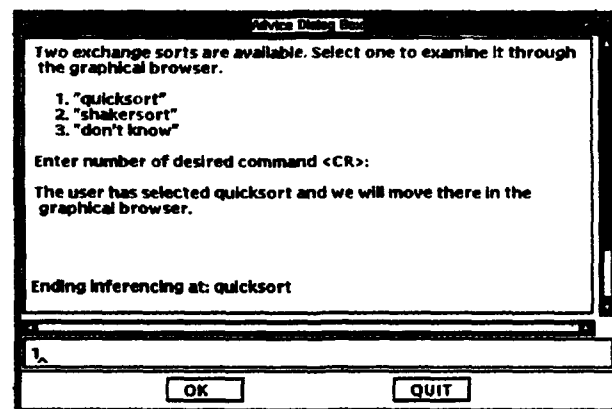


Figure 52: An Example Final Advice Dialog Box

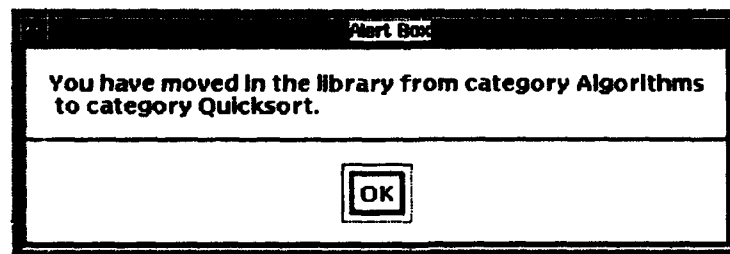


Figure 53: An Alert Box Displayed After an Inferencing Session

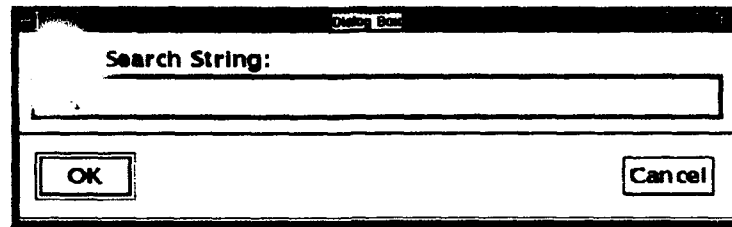


Figure 54: A Search String Dialog Box

### F.3 Scenario 3: Textual Query Mode—Querying Search Mechanisms

In this scenario, the user knows a particular reusable component—a quicksort routine—is needed. Instead of navigating through the library hierarchy, however, the user issues a simple textual query.

#### STEP 1.

Invoke the RLF Graphical Browser from the command line:

```
% Graphical_Browser
```

The Main Menu appears, as shown in Figure 39. Display the menu of libraries by clicking the [Select a Library] button.

#### STEP 2.

Select the desired library. In this case, select the “Sort and Search” library, as shown in Figure 40.

#### STEP 3.

Select the [Search] command from the command bar. The Search String dialog box is displayed, and the user enters a string, as shown in Figure 54.

#### STEP 4.

The results of the textual query are displayed in a control panel. The user selects one of the concepts from the panel, as shown in Figure 55.

#### STEP 5.

The view is centered upon the “example\_quicksort” concept. At this point, the user may then examine and extract the contents of the node, if desired, as in the previous scenarios.

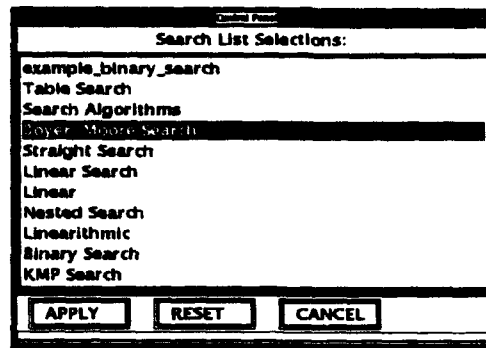


Figure 55: A Search List Selections Menu

## G Appendix: PCTE

In most respects, the PCTE version of this delivery of RLF operates in the same manner as the UNIX version. However, there are differences, and this appendix presents the differences in the PCTE and UNIX versions of RLF and some conventions which can be used to increase portability between versions. This appendix assumes knowledge of PCTE, the Emeralde PCTE product, and the *esh* encapsulated shell. A major assumption for this release of the PCTE RLF is that you install the software into UNIX, then run *esh* scripts in PCTE to install RLF libraries into the PCTE object base, then run the RLF GB from UNIX (since it's not encapsulated – you just use the absolute pathname), and the PCTE version of the RLF GB then accesses the objects in the PCTE object base. For a more detailed discussion of library modeling issues with the PCTE version of the RLF, see the *RLF Modeler's Manual*.

### G.1 Installing the PCTE Version of RLF

This manual assumes the PCTE version of the RLF has been installed and verified, including the tailoring and proper installation of initialization files such as *.profile*. See the *RLF Source Code Release Installation Guide, Version 4.1* or *RLF Binary Release Installation Guide, Version 4.1* for detailed instructions on installing the PCTE version of the RLF.

As a user, you may want to tailor the *.profile* file, the initialization file for the *esh* shell of the PCTE environment. The *.profile* file in the current release looks similar to the following:

```
XAPPLRESDIR=$RLFHOM/pcte/bin; export XAPPLRESDIR
PATH=/rlf.tools:$PATH; export PATH
RLF_LIBRARIES=/Instances4.e; export RLF_LIBRARIES
RLF_PAGER=/usr/local/bin/less; export RLF_PAGER
```

If any of the above pathnames are not appropriate for you, then change them to reflect more suitable values. There are two environment variables that must be set before the RLF GB can be run successfully. They are as follows:

- DISPLAY
- RLF\_LIBRARIES

You can edit the `.profile` file so that these variables are automatically set every time you invoke a new `esh` shell. The `DISPLAY` variable is not set in the above sample `.profile` file, but it could be set there if desired. Note that environment variables *are* inherited by the `esh` from its parent process.

The `XAPPLRESDIR` environment variable contains the location of the `RLF_Browser X` resource file. The `bitmaps` directory should exist as a subdirectory of this location.

The `RLF_PAGER` environment variable must be set to the pathname of a text pager. This is the text pager that the RLF GB will invoke inside an `xterm` window to display text.

The `RLF_LIBRARIES` environment variable is used by the RLF to determine the location of the RLF libraries to be read. An RLF library must be created before the RLF GB can be used to browse that library.

### G.1.1 Starting the PCTE Server

Consult the *Emeraude V12 System Administration Guide* for information on how to start the PCTE server.

### G.1.2 Logging into PCTE

The "standard" method for logging into PCTE is sufficient for using the RLF GB. Consult the *Emeraude V12 System Administration* guide for details.

Before logging in to PCTE, set the `RLFHOME` environment variable, if it hasn't already been set. The value of the `RLFHOME` environment variable should be a pathname to the top of the RLF installation directory hierarchy. If you are unsure what pathname to use for the `RLFHOME` variable, consult your system administrator or the installer of the RLF system. The pathname is probably the same as the location where the release was extracted from the transfer media.

The `RLFHOME` environment variable is used for convenience; you could type the entire absolute pathname instead, but that can become cumbersome.

```
% setenv RLFHOME pathname
```

Assuming your command search path is set properly to find the Emeraude PCTE commands, and assuming you have been set up as valid PCTE user, you should be able to issue the `log` command as follows:

```
% log
```

This should invoke an *esh* shell with a prompt similar to the following:

```
esh$
```

### G.1.3 Creating RLF Libraries in the PCTE Object Base

If RLF libraries have already been installed into your PCTE object base, you may skip this section.

After the *.profile* file has been tailored to your satisfaction it must be installed in the PCTE object base before it can have any effect. The UNIX file is copied into the object base with the PCTE *object\_copy* command as in the following example:

```
esh$ obj_copy -c $RLFHOME/pcte/bin/pcte.profile $PCTE_HOME/.profile
```

Before the RLF GB can be successfully invoked in the PCTE environment, an RLF library must be installed into a PCTE object base. To accomplish this, the following procedures may be used.

Any of the *esh* scripts in the *models* subdirectory of the RLF release may be used to create the associated RLF library in the PCTE object base. The following examples show the creation of the "Animals" RLF library.

Invoke the *esh* script using the full UNIX pathname to the script, with an argument of the pathname to the RLFHOME location:

```
esh$ $RLFHOME/models/animals/BuildAnimalsLib.esh $RLFHOME
```

The script *BuildAnimalsLib.esh* copies the necessary files from UNIX into the PCTE object base and executes the *Lmdl* translator on the *LMDL* script.

### G.1.4 Invoking the PCTE RLF Graphical Browser

After an RLF library has been installed in the PCTE object base, the RLF *Graphical Browser* may be invoked. To invoke the RLF GB from within the PCTE environment, use the full UNIX pathname as in the following example:

```
esh$ $RLFHOME/pcte/bin/Graphical.Browser
```



## G.2 File Naming Restrictions

The Emeraude implementation of PCTE places restrictions on the length of object names and makes assumptions about the use of '.' in object names. The names of files containing assets which are available in an RLF reuse library are restricted to 32 characters in length when using PCTE. These are the files that reside beneath the **Text** subdirectory of any directory where RLF libraries have been constructed. Additionally, the names of files containing reusable assets in the library should not contain the '.' character, since this indicates a special meaning to the Emeraude implementation of PCTE. The convention established by this version of RLF for PCTE is to replace any '.' characters in file names with the underscore character, '\_'. An exception to this convention is the `.rlfrc` start-up file, which the PCTE version of RLF will look for as an entity named `rlfrc.e`.

To increase the similarity in the way libraries are represented in the UNIX and PCTE versions, and to ease transition between versions, the preferred link type of every object in or beneath the directory object where the library was built must be set to ".e". This includes files representing a library's assets and any action scripts which might appear below the **Text** directory. The preferred link type of the directory object indicated by the environment variable, `RLF_LIBRARIES`, also needs to be ".e" so that its subdirectories can be traversed easily.

Library representations built with the PCTE version of RLF also require a directory object named `rlf_tools` to be a first-level subdirectory of the directory object where the library is built. This directory object must also contain two tools named `ascii_file.tool` and `displ_attr.tool`. These tools are required for RLF's default actions to operate correctly.

For examples of library model construction for the PCTE version of RLF, examine the ".esh" versions of the build scripts for the example libraries delivered with the RLF. These scripts are found in each subdirectory of the `models` subdirectory of an RLF installation. These scripts can be modified and reused to help automate the procedures required to build an RLF reuse library with the PCTE version.

**H Appendix: Reporting Problems**

Like most software, especially that of a prototypical nature, the RLF GB may contain unknown bugs (along with some known bugs). If you encounter any problems with this software, or have any suggestions for enhancements, you are encouraged to report them to STARS personnel. The *Version Description Document* included with this release provides instructions for doing this, including information on available Internet electronic mail addresses. Also included in the distribution is a problem report form for formally submitting problem reports.

## I Appendix: References

This section lists a number of resources that are available for further information. The topics are relevant to the use of the RLF GB and include the RLF, the RGB, the X Window System, Ada, and PCTE references.

For more detailed information on the RLF, see the following documents:

For more detailed information on the RLF, see the following documents:

- △ • *RLF Binary Release Installation Guide, Version 4.1*, STARS-UC-05156/012/00; March, 1993.
- △ • *RLF Source Code Installation Guide, Version 4.1*, STARS-UC-05156/014/00; March, 1993.
- △ • *RLF Modeler's Manual, Version 4.1*, STARS-UC-05156/011/00; March, 1993.
- △ • *RLF Administrator's Manual, Version 4.1*, STARS-UC-05156/017/00; March, 1993.
- △ • *RLF Binary Release Version Description Document, Version 4.1*, STARS-UC-05156/016/00; March, 1993.
- △ • *RLF User Tutorial, Version 4.1*, STARS-UC-05156/018/00; March, 1993.
- △ • *RLF Administrator Tutorial, Version 4.1*, STARS-UC-05156/019/00; March, 1993.
- △ • *RLF Modeler Tutorial, Version 4.1*, STARS-UC-05156/020/00; March, 1993.
- R. J. Brachman and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, 9(2) (Spring 1985), pp. 171-216.
- Gordon, D.M, "A Graphical User Interface in Ada for Domain-Specific Reuse Libraries," *TRI-Ada '92 Proceedings*, Orlando, FL; November, 1992.
- K. Wallnau, J. Solderitsch, M. Simos, R. McDowell, K. Cassell, and D. Campbell, "Construction of Knowledge-Based Components and Applications in Ada," *Proceedings of AIDA-88, George Mason University, Fourth Annual Conference on Artificial Intelligence and Ada*, November 1988, pp. 3-1 through 3-21.

For more detailed information on the RGB, see the following documents:

- *RGB 1.0 Version Description Document (VDD)*, STARS-US-020401/001/00
- *RGB 1.0 User's Manual*, STARS-US-020401/002/00

For more detailed information on using X and the *twm* window manager, see the following book:

- *X Window System User's Guide for X11 R3 and R4, Third Edition*; Quercia, Valerie, and O'Reilly, Tim; O'Reilly & Associates, Inc.; Sebastapol, CA; 95472; May 1990.

For more detailed information on Ada/Motif see the following documents:

- *Ada/Motif Release 1.1, The Complete Ada Binding for X11R4 and OSF/Motif 1.1, Users Manual*, Systems Engineering Research Corporation (SERC), Mountain View, CA; Sept. 28, 1992.

For additional information on the Asset Library Open Architecture Framework (ALOAF) and the Common Data Model, see the following document:

- *Asset Library Open Architecture Framework, Version 1.0*; STARS-TC-04041/001/01; 28 February 1992.

For more detailed information on Sun Ada see the following documents:

- *Sun Ada User's Guide*, March, 1992
- *SPARCworks/Ada User's Guide*, March, 1992

For more detailed information on Emeraude PCTE see the following documents:

- *The Emeraude Environment*, GIE Emeraude, 1992; PC6A1, 68, route de Versailles, 788430 Louveciennes, FRANCE.